

NVIDIA Base Command Manager 11

Administrator Manual

Revision: d7928f6

Date: Wed Oct 21 2026

©2026 NVIDIA Corporation & affiliates. All Rights Reserved. This manual or parts thereof may not be reproduced in any form unless permitted by contract or by written permission of NVIDIA Corporation.

Trademarks

Linux is a registered trademark of Linus Torvalds. PathScale is a registered trademark of Cray, Inc. Red Hat and all Red Hat-based trademarks are trademarks or registered trademarks of Red Hat, Inc. SUSE is a registered trademark of SUSE LLC. NVIDIA, CUDA, GPUDirect, HPC SDK, NVIDIA DGX, NVIDIA Nsight, and NVLink are registered trademarks of NVIDIA Corporation. FLEXlm is a registered trademark of Flexera Software, Inc. Green Provisioning is a trademark of Altair Engineering, Inc. All other trademarks are the property of their respective owners.

Rights and Restrictions

All statements, specifications, recommendations, and technical information contained herein are current or planned as of the date of publication of this document. They are reliable as of the time of this writing and are presented without warranty of any kind, expressed or implied. NVIDIA Corporation shall not be liable for technical or editorial errors or omissions which may occur in this document. NVIDIA Corporation shall not be liable for any damages resulting from the use of this document.

Limitation of Liability and Damages Pertaining to NVIDIA Corporation

The NVIDIA Base Command Manager product principally consists of free software that is licensed by the Linux authors free of charge. NVIDIA Corporation shall have no liability nor will NVIDIA Corporation provide any warranty for the NVIDIA Base Command Manager to the extent that is permitted by law. Unless confirmed in writing, the Linux authors and/or third parties provide the program as is without any warranty, either expressed or implied, including, but not limited to, marketability or suitability for a specific purpose. The user of the NVIDIA Base Command Manager product shall accept the full risk for the quality or performance of the product. Should the product malfunction, the costs for repair, service, or correction will be borne by the user of the NVIDIA Base Command Manager product. No copyright owner or third party who has modified or distributed the program as permitted in this license shall be held liable for damages, including general or specific damages, damages caused by side effects or consequential damages, resulting from the use of the program or the un-usability of the program (including, but not limited to, loss of data, incorrect processing of data, losses that must be borne by you or others, or the inability of the program to work together with any other program), even if a copyright owner or third party had been advised about the possibility of such damages unless such copyright owner or third party has signed a writing to the contrary.

Table of Contents

Table of Contents . . . 3
0.1 Quickstart . . . 21
0.2 About This Manual . . . 21
0.3 About The Manuals In General . . . 21
0.4 Getting Administrator-Level Support . . . 22
0.5 Getting Professional Services . . . 22

1 Introduction **23**
1.1 NVIDIA Base Command Manager Functions And Aims . . . 23
1.2 The Scope Of The Administrator Manual (This Manual) . . . 23
1.2.1 Installation . . . 23
1.2.2 Configuration, Management, And Monitoring Via BCM Tools And Applications . . . 24
1.3 Outside The Direct Scope Of The Administrator Manual . . . 25

2 Cluster Management With NVIDIA Base Command Manager **27**
2.1 Concepts . . . 27
2.1.1 Devices . . . 27
2.1.2 Software Images . . . 28
2.1.3 Node Categories . . . 29
2.1.4 Node Groups . . . 30
2.1.5 Roles . . . 30
2.1.6 Configuration Overlay . . . 31
2.2 Modules Environment . . . 31
2.2.1 Adding And Removing Modules . . . 31
2.2.2 Using Local And Shared Modules . . . 32
2.2.3 Setting Up A Default Environment For All Users . . . 32
2.2.4 Creating A Modules Environment Module . . . 33
2.2.5 Lua Modules Environment (Lmod) . . . 34
2.3 Authentication . . . 34
2.3.1 Changing Administrative Passwords On The Cluster . . . 34
2.3.2 Logins Using `ssh` . . . 36
2.3.3 Certificates . . . 36
2.3.4 Profiles . . . 37
2.4 Base View GUI . . . 37
2.4.1 Installing The Cluster Management GUI Service . . . 38
2.4.2 Navigating The Cluster With Base View . . . 40
2.5 Cluster Management Shell . . . 47
2.5.1 Invoking `cmsh` . . . 47
2.5.2 Levels, Modes, Help, And Commands Syntax In `cmsh` . . . 52
2.5.3 Working With Objects . . . 56
2.5.4 Accessing Cluster Settings . . . 68

2.5.5 Advanced cmsh Features . . . 69
2.6 Cluster Management Daemon . . . 82
2.6.1 Managing And Inspecting The Cluster Management Daemon . . . 82
2.6.2 Configuring The Cluster Management Daemon . . . 84
2.6.3 CMDaemon Versions . . . 84
2.6.4 Configuring The Cluster Management Daemon Logging Facilities . . . 85
2.6.5 Configuration File Modification, And The FrozenFile Directive . . . 86
2.6.6 Configuration File Conflicts Between The Standard Distribution And BCM For Generated And Non-Generated Files . . . 87
2.6.7 CMDaemon Lite . . . 87

3 Configuring The Cluster 93
3.1 Main Cluster Configuration Settings . . . 93
3.1.1 Cluster Configuration: Various Cluster-wide Settings . . . 94
3.1.2 Cluster Configuration: Some Network-Related Settings . . . 96
3.1.3 Miscellaneous Settings . . . 98
3.1.4 Limiting The Maximum Number Of Open Files . . . 101
3.2 Network Settings . . . 102
3.2.1 Configuring Networks . . . 103
3.2.2 Adding Networks . . . 108
3.2.3 Changing Network Parameters . . . 108
3.2.4 Tools For Viewing Cluster Connections And Connectivity . . . 122
3.3 Configuring Bridge Interfaces . . . 126
3.4 Configuring VLAN interfaces . . . 127
3.4.1 Configuring A VLAN Interface Using cmsh . . . 127
3.4.2 Configuring A VLAN Interface Using Base View . . . 128
3.5 Configuring Bonded Interfaces . . . 129
3.5.1 Adding A Bonded Interface . . . 129
3.5.2 Single Bonded Interface On A Regular Node . . . 129
3.5.3 Multiple Bonded Interface On A Regular Node . . . 130
3.5.4 Bonded Interfaces On Head Nodes And HA Head Nodes . . . 130
3.5.5 Tagged VLAN On Top Of a Bonded Interface . . . 131
3.5.6 Association Of MAC Address With A Bonded Interface . . . 131
3.5.7 Further Notes On Bonding . . . 131
3.6 Configuring InfiniBand Interfaces . . . 132
3.6.1 Installing Software Packages . . . 132
3.6.2 Mellanox Adapter Settings With mlxconfig . . . 133
3.6.3 Subnet Managers . . . 134
3.6.4 InfiniBand Network Settings . . . 135
3.6.5 Verifying Connectivity . . . 136
3.7 Configuring BMC (IPMI/iLO/DRAC/CIMC/Redfish) Interfaces . . . 137
3.7.1 BMC Network Settings . . . 137
3.7.2 BMC Authentication . . . 139
3.7.3 Auditing Cluster Credentials With The credentialsoverview Command . . . 141
3.7.4 Interfaces Settings . . . 142
3.7.5 Identification With A BMC . . . 143
3.7.6 Identification At A Lower Level, Directly With ipmitool . . . 143

3.7.7 Identification At A Higher Level, With The `identify` Command Of Device Mode 143
3.7.8 BMC Event Log . . . 144
3.7.9 BMC Reset . . . 144
3.7.10 BMC Port Queries . . . 144
3.7.11 Leak Detection . . . 145
3.8 Configuring Switches And PDUs . . . 146
3.8.1 Configuring With The Manufacturer's Configuration Interface . . . 146
3.8.2 Configuring SNMP . . . 147
3.9 Configuring Cumulus Switches . . . 149
3.9.1 Cumulus Switches Access Configuration, Initialization And Network Device Discovery . . . 150
3.9.2 Custom Service Setups For Cumulus Linux . . . 152
3.9.3 Uplink Ports . . . 157
3.9.4 The `showport` MAC Address to Port Matching Tool . . . 158
3.9.5 Disabling Port Detection . . . 159
3.9.6 The `switchoverview` Command . . . 160
3.9.7 The `finddevicebyswitchport` Command . . . 160
3.9.8 The `setmacviaswitchport` Command . . . 161
3.9.9 The `checkfactorydefaultpassword` Command . . . 161
3.10 Additional `device` Mode Node Commands And Submodes . . . 161
3.10.1 The `nodeoverview` Command . . . 162
3.10.2 The `additionalnodeinformation` Command . . . 162
3.10.3 The `minimalconfig` Command . . . 163
3.10.4 The `checkremotemount` Command . . . 163
3.10.5 The `proxysettings` Submode . . . 163
3.10.6 The `selinuxsettings` Submode . . . 164
3.10.7 The `remotedaemon` Command . . . 164
3.11 Configuring The NetQ Network Management System . . . 165
3.12 Configuring UFM . . . 165
3.13 Disk Layouts: Disked, Semi-Diskless, And Diskless Node Configuration . . . 166
3.13.1 Disk Layouts . . . 166
3.13.2 Disk Layout Assertions . . . 166
3.13.3 Changing Disk Layouts . . . 167
3.13.4 Changing A Disk Layout Using Base View . . . 167
3.13.5 Changing A Disk Layout Using `cmsh` . . . 168
3.14 Configuring NFS Volume Exports And Mounts . . . 169
3.14.1 Exporting A Filesystem Using Base View And `cmsh` . . . 171
3.14.2 Mounting A Filesystem Using Base View And `cmsh` . . . 174
3.14.3 Mounting A Filesystem Subtree For A Diskless Node Over NFS . . . 177
3.14.4 Configuring NFS Volume Exports And Mounts Over RDMA With OFED Drivers . 179
3.15 Managing And Configuring Services . . . 180
3.15.1 Why Use The Cluster Manager For Services? . . . 180
3.15.2 Managing And Configuring Services—Examples . . . 181
3.16 Managing And Configuring A Rack . . . 188
3.16.1 Racks . . . 188
3.16.2 Assigning Devices To A Rack . . . 192

3.16.3 Assigning Devices To A Chassis . . . 194
3.16.4 The chassisposition Submode . . . 196
3.16.5 Power Management Of Racks . . . 196
3.17 Configuring GPU Settings . . . 196
3.17.1 GPUs And GPU Units . . . 196
3.17.2 Configuring GPU Settings . . . 197
3.17.3 MIG Configuration . . . 204
3.18 Configuring Sampling From A Prometheus Exporter . . . 213
3.19 Configuring Custom Scripts . . . 213
3.19.1 custompowerscript . . . 214
3.19.2 custompingscript . . . 214
3.19.3 customremoteconsolescript . . . 214
3.19.4 sysinfo Custom Scripts . . . 215
3.20 Cluster Configuration Without Execution By CMDaemon . . . 218
3.20.1 Cluster Configuration: The Bigger Picture . . . 218
3.20.2 Making Nodes Function Differently By Image . . . 219
3.20.3 Making All Nodes Function Differently From Normal Cluster Behavior With FrozenFile . . . 221
3.20.4 Adding Functionality To Nodes Via An initialize Or finalize Script . . . 222
3.20.5 Examples Of Configuring Nodes With Or Without CMDaemon . . . 222
3.21 Saving A Backup Of Configuration Files With versionconfigfiles . . . 223

4 Power Management 225
4.1 Configuring Power Parameters . . . 225
4.1.1 PDU-based Power Control . . . 226
4.1.2 IPMI-Based Power Control . . . 229
4.1.3 Combining PDU- and IPMI-Based Power Control . . . 229
4.1.4 Custom Power Control . . . 230
4.1.5 Hewlett Packard iLO-Based Power Control . . . 231
4.1.6 Dell drac-based Power Control . . . 231
4.1.7 Redfish-Based and CIMC-Based Power Control . . . 231
4.2 Power Operations . . . 231
4.2.1 Power Operations Overview . . . 231
4.2.2 Power Operations With Base View . . . 232
4.2.3 Power Operations Through cmsh . . . 233
4.3 Power Infrastructure Overview . . . 237
4.3.1 PDU Overview . . . 237
4.3.2 CDU Overview . . . 237
4.3.3 Power Shelf Overview . . . 237
4.4 Monitoring Power . . . 238
4.5 Switch Configuration To Survive Other Device Power Downs . . . 238

5 Node Provisioning 239
5.1 Before The Kernel Loads . . . 239
5.1.1 PXE Booting . . . 239
5.1.2 iPXE Booting From A Disk Drive . . . 242
5.1.3 iPXE Booting Using InfiniBand . . . 242

5.1.4 Using PXE To Boot From The Drive . . . 243
5.1.5 Network Booting Without PXE On The ARMv8 Architecture . . . 243
5.1.6 Network Booting Protocol . . . 243
5.1.7 The Boot Role . . . 244
5.2 Provisioning Nodes . . . 244
5.2.1 Provisioning Nodes: Configuration Settings . . . 244
5.2.2 Provisioning Nodes: Role Setup With `cmsh` . . . 245
5.2.3 Provisioning Nodes: Role Setup With Base View . . . 247
5.2.4 Provisioning Nodes: Housekeeping . . . 249
5.3 The Kernel Image, Ramdisk And Kernel Modules . . . 256
5.3.1 Booting To A "Good State" Software Image . . . 256
5.3.2 Selecting Kernel Driver Modules To Load Onto Nodes . . . 256
5.3.3 InfiniBand Provisioning . . . 258
5.3.4 VLAN Provisioning . . . 260
5.4 Node-Installer . . . 261
5.4.1 Requesting A Node Certificate . . . 262
5.4.2 Deciding Or Selecting Node Configuration . . . 264
5.4.3 Starting Up All Network Interfaces . . . 275
5.4.4 Determining Install-mode Type And Execution Mode . . . 276
5.4.5 Running Initialize Scripts . . . 281
5.4.6 Checking Partitions, RAID Configuration, Mounting Filesystems . . . 281
5.4.7 Synchronizing The Local Drive With The Software Image . . . 282
5.4.8 Writing Network Configuration Files . . . 286
5.4.9 Creating A Local `/etc/fstab` File . . . 287
5.4.10 Booting From The Local Hard Drive . . . 287
5.4.11 Running Finalize Scripts . . . 289
5.4.12 Unloading Specific Drivers . . . 290
5.4.13 Switching To The Local `init` Process . . . 290
5.5 Node States . . . 290
5.5.1 Node States Icons In Base View . . . 290
5.5.2 Node States Shown In `cmsh` . . . 291
5.5.3 Node States Indicating Regular Start Up . . . 291
5.5.4 Node States That May Indicate Problems . . . 292
5.6 Updating Running Nodes . . . 294
5.6.1 Updating Running Nodes: Configuration With `excludelistupdate` . . . 295
5.6.2 Updating Running Nodes: With `cmsh` Using `imageupdate` . . . 302
5.6.3 Updating Running Nodes: With Base View Using the `Update node` Option . . . 302
5.6.4 Updating Running Nodes: Considerations . . . 302
5.7 Adding New Nodes . . . 303
5.7.1 Adding New Nodes With `cmsh` And Base View Add Functions . . . 303
5.7.2 Adding New Nodes With The Node Creation Wizard . . . 303
5.8 Troubleshooting The Node Boot Process . . . 309
5.8.1 Node Fails To PXE Boot . . . 309
5.8.2 Node-installer Logging . . . 313
5.8.3 Provisioning Logging . . . 314
5.8.4 Ramdisk Fails During Loading Or Sometime Later . . . 314

5.8.5 Ramdisk Fails With `No space left on device` . . . 315
5.8.6 Ramdisk Cannot Start Network . . . 315
5.8.7 Node-Installer Cannot Create Disk Layout . . . 316
5.8.8 Node-Installer Cannot Start BMC (IPMI/iLO) Interface . . . 319

6 User Management **325**
6.1 Managing Users And Groups With Base View . . . 325
6.2 Managing Users And Groups With `cmsh` . . . 327
6.2.1 Adding A User . . . 327
6.2.2 Saving The Modified State . . . 328
6.2.3 Editing Properties Of Users And Groups . . . 329
6.2.4 Reverting To The Unmodified State . . . 332
6.2.5 Removing A User . . . 332
6.3 Using An External LDAP Server . . . 333
6.3.1 External LDAP Server Replication . . . 335
6.3.2 High Availability . . . 337
6.4 Keycloak Integration . . . 338
6.4.1 Prerequisites . . . 338
6.4.2 Installing Keycloak And Federating LDAP . . . 339
6.4.3 Verifying The Installation . . . 339
6.4.4 Related Commands . . . 339
6.4.5 NVIDIA Mission Control Integration . . . 339
6.5 Tokens And Profiles . . . 339
6.5.1 Modifying Profiles . . . 341
6.5.2 Creation Of Custom Certificates With Profiles, For Users Managed By BCM's Internal LDAP . . . 342
6.5.3 Creation Of Custom Certificates With Profiles, For Users Managed By An External LDAP . . . 346
6.5.4 Logging The Actions Of CMDaemon Users . . . 347
6.5.5 Creation Of Certificates For Nodes With `cm-component-certificate` . . . 347

7 Workload Management **351**
7.1 Workload Managers Choices . . . 351
7.2 Forcing Jobs To Run In A Workload Management System . . . 352
7.2.1 Disallowing User Logins To Regular Nodes Via `cmsh` . . . 352
7.2.2 Disallowing User Logins To Regular Nodes Via Base View . . . 353
7.2.3 Disallowing Other User Processes Outside Of Workload Manager User Processes . . . 353
7.2.4 High Availability By Workload Managers . . . 354
7.3 Installation Of Workload Managers . . . 357
7.3.1 Running `cm-wlm-setup` In CLI Mode . . . 357
7.3.2 Running `cm-wlm-setup` As A TUI . . . 360
7.3.3 Installation And Configuration Of Enroot And Pyxis With Slurm To Run Containerized Jobs . . . 365
7.3.4 Prolog And Epilog Scripts . . . 371
7.4 Enabling, Disabling, And Monitoring Workload Managers . . . 374
7.4.1 Enabling And Disabling A WLM With Base View . . . 374
7.4.2 Enabling And Disabling A Workload Manager With `cmsh` . . . 378

7.4.3 Monitoring The State Of Workload Manager Services . . . 383
7.5 Configuring And Running Individual Workload Managers . . . 387
7.5.1 Configuring And Running Slurm . . . 387
7.5.2 Installing, Configuring, And Running LSF . . . 431
7.6 Using Base View With Workload Management . . . 439
7.6.1 Jobs Display And Handling In Base View . . . 439
7.6.2 Queues Display And Handling In Base View . . . 441
7.7 Using `cmsh` With Workload Management . . . 442
7.7.1 The `activeservers` Command . . . 444
7.7.2 The `jobs` Submode In `cmsh` . . . 445
7.7.3 Job Queue Display And Handling In `cmsh`: `jobqueue` Mode . . . 453
7.7.4 Nodes Drainage Status And Handling In `cmsh` . . . 454
7.8 Examples Of Workload Management Assignment . . . 456
7.8.1 Setting Up A New Category And A New Queue For It . . . 456
7.8.2 Setting Up A Prejob Or Postjob Check . . . 459
7.9 Power Saving With `cm-scale` . . . 461
7.10 Cgroups . . . 461
7.10.1 Cgroups Settings For Workload Managers . . . 462
7.11 Custom Node Parameters . . . 465

8 NVIDIA Base Command Manager Auto Scaler 467
8.1 Introduction . . . 467
8.1.1 Use Cases . . . 467
8.1.2 Resource Constraints . . . 468
8.1.3 Setup . . . 472
8.1.4 Workload Roles Assignment Limitations Per Node With `cm-scale` . . . 481
8.2 Configuration . . . 481
8.2.1 The ScaleServer Role . . . 481
8.2.2 Resource Providers . . . 483
8.2.3 Time Quanta Optimization . . . 486
8.2.4 Fairsharing Priority Calculation And Node Management . . . 488
8.2.5 Engines . . . 489
8.2.6 Trackers . . . 490
8.3 Examples Of `cm-scale` Use . . . 497
8.3.1 Simple Static Node Provider Usage Example . . . 497
8.3.2 Simple Dynamic Node Provider Usage Example . . . 500
8.4 Further `cm-scale` Configuration And Examples . . . 506
8.4.1 Dynamic Nodes Re-purposing . . . 506
8.4.2 Pending Reasons . . . 507
8.4.3 Locations . . . 508
8.4.4 Azure Storage Accounts Assignment . . . 509
8.4.5 Uptake of HPC Jobs By Particular Types Of Nodes . . . 510
8.4.6 How To Exclude Unused Nodes From Being Stopped . . . 511
8.4.7 Prolog And Epilog Scripts With Auto Scaler . . . 512
8.4.8 Queue Node Placeholders . . . 513
8.4.9 Auto Scaling A Job On-premises To A Workload Manager And Kubernetes . . . 514
8.4.10 AWS Spot Instances And Availability Zones . . . 515

8.4.11 Auto Scaler Statistics . . . 516

9 Post-installation Software Management 519
9.1 NVIDIA Base Command Manager Packages, Their Naming Convention And Version . . 521
9.1.1 The `packages` Command . . . 523
9.1.2 BCM Package Point Release Versions And The `cm-package-release-info` Command . . . 524
9.2 Managing Packages On The Head Node . . . 526
9.2.1 Managing RPM Or .deb Packages On The Head Node . . . 526
9.2.2 Installation Of Packages On The Head Node That Are Not .deb And Not .rpm Packages . . . 528
9.3 Kernel Management On A Head Node Or Image . . . 529
9.3.1 Installing A Standard Distribution Kernel Into An Image Or On A Head Node . . 529
9.3.2 Excluding Kernels And Other Packages From Updates . . . 530
9.3.3 Updating A Kernel In A Software Image . . . 531
9.3.4 Setting Kernel Options For Software Images . . . 532
9.3.5 Kernel Driver Modules . . . 533
9.4 Managing A Package In A Software Image And Running It On Nodes . . . 535
9.4.1 Installing From Head Into The Image: Changing The Root Directory Into Which The Packages Are Deployed . . . 535
9.4.2 Installing From Head Into The Image: Updating The Node . . . 537
9.4.3 Installing From Head Into The Image: Possible Issues When Using `rpm --root`, `yum --installroot` Or `chroot` . . . 537
9.4.4 Managing A Package In The Node-Installer Image . . . 538
9.5 Managing Non-RPM Software In A Software Image And Running It On Nodes . . . 539
9.5.1 Managing The Software Directly On An Image . . . 539
9.5.2 Managing The Software Directly On A Node, Then Syncing Node-To-Image . . . 540
9.6 Creating A Custom Software Image . . . 543
9.6.1 Creating A Base Distribution Archive From A Base Host . . . 544
9.6.2 Creating The Software Image With `cm-create-image` . . . 546
9.6.3 Configuring Local Repositories For Linux Distributions, And For The BCM Package Repository, For A Software Image . . . 550
9.6.4 Creating A Custom Image From The Local Repository . . . 554
9.7 Creating Images For Other Distributions And Architectures (Multidistro And Multiarch) 554
9.7.1 The `cm-image` Tool . . . 555
9.7.2 Multidistro Examples: Provisioning From Rocky 8 Head Node To Ubuntu 24.04 Regular Nodes . . . 559
9.7.3 Multiarch Example: Creating An Image From A Centos 8 Head Node For ARMv8 Architecture Regular Nodes . . . 560

10 Monitoring: Monitoring Cluster Devices 565
10.1 A Basic Monitoring Example And Action . . . 565
10.1.1 Synopsis Of Basic Monitoring Example . . . 565
10.1.2 Before Using The Basic Monitoring Example—Setting Up The Pieces . . . 566
10.1.3 Using The Basic Monitoring Example . . . 567
10.2 Monitoring Concepts And Definitions . . . 570
10.2.1 Measurables . . . 570

10.2.2 Enummetrics . . . 573
10.2.3 Metrics . . . 574
10.2.4 Health Check . . . 576
10.2.5 Trigger . . . 577
10.2.6 Action . . . 577
10.2.7 Severity . . . 578
10.2.8 AlertLevel . . . 578
10.2.9 Flapping . . . 578
10.2.10 Data Producer . . . 579
10.2.11 Conceptual Overview: The Main Monitoring Interfaces Of Base View . . . 582
10.3 Monitoring Visualization With Base View . . . 583
10.3.1 The Monitoring Window . . . 583
10.4 Monitoring Configuration With Base View . . . 585
10.4.1 Monitoring Configuration: Data Producers . . . 586
10.4.2 Monitoring Configuration: Measurables . . . 588
10.4.3 Monitoring Configuration: Consolidators . . . 590
10.4.4 Monitoring Configuration: Actions . . . 595
10.4.5 Monitoring Configuration: Triggers . . . 597
10.4.6 Monitoring Configuration: Health status . . . 601
10.4.7 Monitoring Configuration: All Health Checks . . . 602
10.4.8 Monitoring Configuration: Standalone Monitored Entities . . . 603
10.4.9 Monitoring Configuration: PromQL Queries . . . 603
10.5 The `monitoring` Mode Of `cmsh` . . . 604
10.5.1 The `action` Submode . . . 605
10.5.2 The `consolidator` Submode . . . 607
10.5.3 The `measurable` Submode . . . 609
10.5.4 The `setup` Submode . . . 614
10.5.5 The `standalone` Submode . . . 624
10.5.6 The `trigger` Submode . . . 624
10.6 Obtaining Monitoring Data Values . . . 627
10.6.1 Getting The List Of Measurables For An Entity: The `measurables`, `metrics`, `healthchecks` And `enummetrics` Commands . . . 627
10.6.2 On-demand Metric Sampling And Health Checks . . . 628
10.6.3 The Latest Data And Counter Values—The `latest*data` And `latestmetriccounters` Commands . . . 631
10.6.4 Data Values Over A Period—The `dumpmonitoringdata` Command . . . 634
10.6.5 Monitoring Data Health Overview—The `healthoverview` Command . . . 645
10.6.6 Monitoring Data About The Monitoring Action History—The `monitoringactionhistory` Command . . . 645
10.6.7 Monitoring Data About The Monitoring System—The `monitoringinfo` Command 645
10.6.8 Dropping Monitoring Data With The `monitoringdrop` Command . . . 647
10.6.9 Monitoring Suspension And Resumption—The `monitoringsuspend` And `monitoringresume` Commands . . . 648
10.6.10 Monitoring Pickup Intervals . . . 649
10.7 Offloaded Monitoring . . . 651
10.7.1 Why Offloaded Monitoring? . . . 651

10.7.2 Implementing Offloaded Monitoring . . . 652
10.7.3 Background Details . . . 652
10.7.4 Examining Offloaded Monitoring With monitoringoffloadinformation . . . 656
10.8 The User Portal . . . 657
10.8.1 Accessing The User Portal . . . 657
10.8.2 Setting A Common Username/Password For The User Portal . . . 658
10.8.3 User Portal Access . . . 658
10.8.4 User Portal Home Page . . . 659
10.9 Cloud Job Tagging . . . 659
10.10 Event Viewer . . . 660
10.10.1 Viewing Events In Base View . . . 660
10.10.2 Viewing Events In cmsh . . . 661
10.10.3 Using The Event Bucket From The Shell For Events And For Tagging Device States 663
10.10.4 InfoMessages . . . 663
10.11 Monitoring Location With GNSS . . . 665
10.12 Monitoring Report Queries . . . 666
10.12.1 Monitoring Report Queries In cmsh . . . 666
10.13 Monitoring With nvsm . . . 668
10.14 Monitoring With nvdebug . . . 672

11 Monitoring: Job Monitoring 675
11.1 Job Metrics Introduction . . . 675
11.2 Job Metrics With Cgroups . . . 675
11.3 Job Information Retention . . . 676
11.4 Job Metrics Sampling Configuration . . . 677
11.4.1 The Job Metrics Collection Processing Mechanism . . . 679
11.5 Job Monitoring In cmsh . . . 680

12 Monitoring: Job Accounting 685
12.1 Introduction . . . 685
12.2 Labeled Entities . . . 685
12.2.1 Dataproducers For Labeled Entities . . . 686
12.2.2 PromQL And Labeled Entities . . . 686
12.2.3 Job IDs And Labeled Entities . . . 686
12.2.4 Measurables And Labeled Entities . . . 686
12.3 PromQL Queries . . . 687
12.3.1 The Default PromQL Queries... . . . 687
12.3.2 ...And A Short Description Of Them . . . 689
12.3.3 Modifying The Default PromQL Query Properties . . . 693
12.3.4 An Example PromQL Query, Properties, And Disassembly . . . 694
12.3.5 Aside: Getting Raw Values For A Prometheus Class Metric . . . 695
12.3.6 ...An Example PromQL Query, Properties, And Disassembly (Continued) . . . 696
12.3.7 Time Specification With PromQL . . . 697
12.4 Parameterized PromQL Queries . . . 698
12.4.1 Two Job GPU Metrics Used In PromQL Queries . . . 700
12.5 Job Accounting In Base View . . . 701
12.5.1 Management And Use Of The Accounting Panel . . . 703

12.6 Access Control For Workload Accounting And Reporting . . . 710
12.6.1 Defining Project Managers Using Internal User Management . . . 710
12.6.2 Defining Project Managers Using External User Management . . . 711
12.7 Drilldown Queries For Workload Accounting And Reporting . . . 712
12.7.1 The drilldownoverview Command . . . 713
12.8 The grid Command For Job Accounting . . . 715
12.8.1 The grid Command Help Text . . . 715
12.8.2 Some grid Command Examples . . . 715
12.8.3 The grid Command Time Specification . . . 717

13 Monitoring: Job Chargeback 719
13.1 Introduction . . . 719
13.1.1 The Word "Chargeback" . . . 719
13.1.2 Comparison Of Job Chargeback Monitoring Measurement With Other Monitoring Measurements . . . 719
13.2 Job Chargeback Measurement . . . 720
13.2.1 Predefined Job Chargebacks . . . 720
13.2.2 Setting A Custom Job Chargeback . . . 721
13.2.3 The report And request Commands . . . 722
13.3 Job Chargeback Background Information . . . 727

14 Day-to-day Administration 729
14.1 Parallel Shells: pdsh And pexec . . . 729
14.1.1 pdsh In The OS Shell . . . 730
14.1.2 pexec In cmsh . . . 733
14.1.3 pexec In Base View . . . 734
14.1.4 Using The -j|--join Option Of pexec In cmsh . . . 734
14.1.5 Other Parallel Commands . . . 735
14.2 Getting Support With BCM Issues, And Notifications For Release Updates . . . 735
14.2.1 The Support Portal For BCM . . . 736
14.2.2 Reporting Cluster Manager Diagnostics With cm-diagnose . . . 737
14.2.3 Getting Notified About Updates . . . 738
14.3 Backups . . . 738
14.3.1 Cluster Installation Backup . . . 738
14.3.2 Local Database And Data Backups And Restoration . . . 739
14.4 Revision Control For Images . . . 741
14.4.1 Btrfs: The Concept And Why It Works Well In Revision Control For Images . . . 741
14.4.2 Btrfs Availability And Distribution Support . . . 742
14.4.3 Installing Btrfs To Work With Revision Control Of Images In BCM . . . 742
14.4.4 Using cmsh For Revision Control Of Images . . . 744
14.5 BIOS And Firmware Management . . . 747
14.5.1 Introduction . . . 747
14.5.2 BIOS Management With BCM JSON Configuration Templates In Redfish . . . 747
14.5.3 Updating BIOS And Firmware Versions . . . 753
14.6 Hardware Match Check With The hardware-profile Data Producer . . . 766
14.7 Serial Over LAN Console Access . . . 767
14.7.1 Background Notes On Serial Console And SOL . . . 768

14.7.2 SOL Console Configuration With Base View . . . 769
14.7.3 SOL Console Configuration And Access With cmsh . . . 770
14.7.4 The conman Serial Console Logger And Viewer . . . 771
14.8 Managing Raw Monitoring Data . . . 774
14.8.1 Monitoring Subsystem Disk Usage With The monitoringinfo --storage Option 774
14.8.2 Estimating The Required Size Of The Storage Device . . . 774
14.8.3 Moving Monitoring Data Elsewhere . . . 775
14.8.4 Reducing Monitoring Data By Reducing Samples . . . 775
14.8.5 Deleting All Monitoring Data . . . 775
14.9 Node Replacement . . . 777
14.10 Ansible And NVIDIA Base Command Manager . . . 777
14.10.1 An Overview Of Ansible . . . 777
14.10.2 A Simple Playbook Example . . . 778
14.10.3 An Intermediate Playbook Example: Setting Up A Cluster For Demonstration Purposes . . . 780
14.10.4 A More Complicated Playbook Example: Creating An Edge Site And Related Properties . . . 782
14.11 Querying Hardware Information . . . 788
14.11.1 Hardware Inventory Information: hardwareinventoryinfo . . . 788
14.11.2 PCI Device Information: lspci . . . 789
14.11.3 EEPROM Information: lseeprom . . . 789
14.11.4 Accelerator Count: accelerators . . . 790
14.12 The cm-mysql-setup Utility . . . 790
14.12.1 Introduction . . . 790
14.12.2 Installation Through The Wizard . . . 790
14.12.3 Installation From CLI . . . 792
14.12.4 Removal From CLI . . . 795
14.12.5 Clean Up . . . 795
14.12.6 Maintenance . . . 796

15 High Availability **799**
15.0 Introduction . . . 799
15.0.1 Why Have High Availability? . . . 799
15.0.2 High Availability—For What Nodes? . . . 799
15.0.3 High Availability Usually Uses Shared Storage . . . 800
15.0.4 Organization Of This Chapter . . . 800
15.1 HA Concepts . . . 800
15.1.1 Primary, Secondary, Active, Passive . . . 800
15.1.2 Monitoring The Active Head Node, Initiating Failover . . . 800
15.1.3 Services In BCM HA Setups . . . 801
15.1.4 Failover Network Topology . . . 802
15.1.5 Shared Storage . . . 804
15.1.6 Guaranteeing One Active Head At All Times . . . 805
15.1.7 Automatic Vs Manual Failover . . . 806
15.1.8 HA And Cloud Nodes . . . 807
15.1.9 HA Using Virtual Head Nodes . . . 807
15.2 HA Setup Procedure Using cmha-setup . . . 807

15.2.1 Preparation . . . 808
15.2.2 Failover Cloning (Replacing A Passive Head) . . . 811
15.2.3 Shared Storage Setup . . . 814
15.2.4 Automated Failover And Relevant Testing . . . 814
15.3 Running cmha-setup Without ncurses, Using An XML Specification . . . 816
15.3.1 Why Run It Without ncurses? . . . 816
15.3.2 The Syntax Of cmha-setup Without ncurses . . . 816
15.3.3 Example cmha-setup Run Without ncurses . . . 816
15.4 Managing HA . . . 817
15.4.1 Changing An Existing Failover Configuration . . . 817
15.4.2 cmha Utility . . . 817
15.4.3 States . . . 821
15.4.4 Failover Action Decisions . . . 822
15.4.5 Keeping Head Nodes In Sync . . . 823
15.4.6 High Availability Parameters . . . 825
15.4.7 Viewing Failover Using Base View . . . 827
15.4.8 Re-cloning A Head Node . . . 827
15.5 HA For Regular Nodes And Edge Director Nodes . . . 828
15.5.1 Why Have HA On Non-Head Nodes? . . . 829
15.5.2 Comparing HA For Head Nodes, Regular Nodes And Edge Director Nodes . . . 829
15.5.3 Setting Up A Regular Node HA Service . . . 830
15.5.4 The Sequence Of Events When Making Another HA Regular Node Active . . . 834
15.6 HA And Workload Manager Jobs . . . 834

16 The Jupyter Notebook Environment Integration 837
16.1 Introduction . . . 837
16.2 Jupyter Environment Installation . . . 838
16.2.1 Jupyter Setup . . . 839
16.2.2 Jupyter Architecture . . . 840
16.2.3 Verifying Jupyter Installation . . . 841
16.2.4 Login Configuration . . . 842
16.2.5 JupyterHub Screen After Login . . . 843
16.3 Jupyter Notebook Examples . . . 844
16.4 Jupyter Kernels . . . 845
16.4.1 Jupyter Kernel Provisioning Kernels . . . 847
16.4.2 Tunables For Kernel Provisioners . . . 848
16.5 Jupyter Kernel Creator Extension . . . 850
16.5.1 BCM Predefined Kernel Templates . . . 851
16.5.2 Jupyter Kernel Starter . . . 855
16.5.3 Running Jupyter Kernels With Two Factor Authentication . . . 856
16.5.4 Running Jupyter Kernels With Kubernetes . . . 857
16.5.5 Running Jupyter Kernels Based On NGC Containers . . . 857
16.5.6 Running Jupyter Kernels With Workload Managers . . . 860
16.6 Jupyter Kernel Creator Extension Customization . . . 861
16.6.1 Kernel Template Parameters Definition . . . 861
16.6.2 Kernel Template Parameters Usage . . . 864
16.6.3 Filtering Out Irrelevant Templates From The Interface For Users . . . 865

16.7 Jupyter VNC Extension . . . 866
16.7.1 What Is Jupyter VNC Extension About? . . . 866
16.7.2 Enabling User Lingering . . . 867
16.7.3 Starting A VNC Session With The Jupyter VNC Extension . . . 867
16.7.4 Running Examples And Applications In The VNC Session With The Jupyter VNC Extension . . . 869
16.8 Jupyter WLM Magic Extension . . . 870
16.9 Jupyter Kubernetes Operators Manager . . . 872
16.9.1 Overview Tab . . . 873
16.9.2 Jupyter Kernel Overview Tab . . . 874
16.9.3 Jobs Tab . . . 874
16.9.4 Pods Tab . . . 876
16.9.5 PVCs Tab . . . 877
16.9.6 PSQL Tab . . . 879
16.9.7 Spark Tab . . . 881
16.9.8 Events Tab . . . 889
16.10Jupyter Environment Removal . . . 890

A Generated Files **891**
A.1 System Configuration Files Created Or Modified By CMDaemon On Head Nodes . . . 891
A.2 System Configuration Files Created Or Modified Directly On The Node . . . 895
A.2.1 Options To `filewriteinfo` . . . 895
A.2.2 Files Created On Regular Nodes By CMDaemon . . . 896
A.2.3 Files Created On Regular Nodes By The Node-Installer . . . 897
A.3 Files Not Generated, But Installed In RHEL And Derivatives . . . 898

B Bright Computing Public Key **903**

C CMDaemon Configuration File Directives **905**

D Disk Partitioning And RAID Configuration **939**
D.1 Structure Of Partitioning Definition—The Global Partitioning XML Schema Definition File 939
D.2 Structure Of Hardware RAID Definition—The Hardware RAID XML Schema Definition File . . . 947
D.3 Example: Default Node Partitioning . . . 950
D.3.1 Multiple Disks And Devices . . . 951
D.3.2 Persistent Naming For NVMe Disks . . . 951
D.3.3 By-Path Device References And Cloud Devices . . . 952
D.3.4 Partition Sizing And Filesystem Types . . . 952
D.3.5 Mount Options . . . 953
D.4 Example: Hardware RAID Configuration . . . 953
D.4.1 RAID level 0 And RAID 10 Example . . . 953
D.5 Example: Software RAID . . . 955
D.6 Example: Software RAID With Swap . . . 956
D.7 Example: Logical Volume Manager . . . 957
D.8 Example: Logical Volume Manager With RAID 1 . . . 958
D.9 Example: Diskless . . . 959

D.10 Example: Semi-diskless . . . 960
D.11 Example: Preventing Accidental Data Loss . . . 960
D.12 Example: Using Custom Assertions . . . 962
D.13 Example: Software RAID1 With One Big Partition . . . 963
D.14 Example: Software RAID5 With One Big Partition . . . 964
D.15 Example: Software RAID1 With Standard Partitioning . . . 966
D.16 Example: Software RAID5 With Standard Partitioning . . . 969
D.17 Example: LUKS Disk Encryption With Standard Partitioning . . . 972
D.17.1 Introduction . . . 972
D.17.2 Node Provisioned Over The Network: Encrypted Partition XML Example . . . 972
D.17.3 Standalone Node: Encrypted Partition XML Example . . . 975
D.17.4 Changing A Passphrase On An Encrypted Node . . . 976

E Example `initialize` And `finalize` Scripts **977**
E.1 When Are They Used? . . . 977
E.2 Accessing From Base View And `cmsh` . . . 977
E.3 Environment Variables Available To `initialize` And `finalize` Scripts . . . 978
E.4 Using Environment Variables Stored In Multiple Variables . . . 981
E.5 Storing A Configuration To A Filesystem . . . 982
E.5.1 Storing With Initialize Scripts . . . 982
E.5.2 Ways Of Writing A Finalize Script To Configure The Destination Nodes . . . 982
E.5.3 Restricting The Script To Nodes Or Node Categories . . . 985

F Workload Managers Quick Reference **987**
F.1 Slurm . . . 987

G Metrics, Health Checks, Enummetrics, And Actions **991**
G.1 Metrics And Their Parameters . . . 991
G.1.1 Regular Metrics . . . 992
G.1.2 NFS Metrics . . . 999
G.1.3 InfiniBand Metrics . . . 1001
G.1.4 Monitoring System Metrics . . . 1002
G.1.5 CPU Metrics Sampled By The CPUSampler And GPUSampler . . . 1005
G.1.6 GPU Metrics . . . 1005
G.1.7 GPU Profiling Metrics . . . 1007
G.1.8 Job Metrics . . . 1008
G.1.9 IPMI Metrics . . . 1015
G.1.10 Redfish Metrics . . . 1016
G.1.11 SMART Metrics . . . 1017
G.1.12 Prometheus Metrics . . . 1018
G.1.13 NetQ Metrics . . . 1020
G.1.14 Kubernetes Metrics . . . 1021
G.1.15 Parameters For Metrics . . . 1031
G.2 Health Checks And Their Parameters . . . 1034
G.2.1 Regular Health Checks . . . 1035
G.2.2 GPU Health Checks . . . 1041
G.2.3 Redfish Health Checks . . . 1042

G.2.4 NetQ Health Checks . . . 1043
G.2.5 Parameters For Health Checks . . . 1043
G.3 Enummetrics . . . 1044
G.4 Actions And Their Parameters . . . 1046
G.4.1 Actions . . . 1046
G.4.2 Parameters For A Monitoring Action . . . 1046

H Workload Manager Configuration Files Updated By CMDaemon 1049
H.1 Slurm . . . 1049
H.2 LSF . . . 1049

I Changing The LDAP Password 1051
I.1 Setting A New Password For The LDAP Server . . . 1051
I.2 Setting The New Password In `cmd.conf` . . . 1051
I.3 Checking LDAP Access . . . 1052

J Tokens 1053

K Understanding Consolidation 1069
K.1 Introduction . . . 1069
K.2 What Is Consolidation? . . . 1069
K.3 Raw Data And Consolidation . . . 1069
K.4 A Demonstration Of The Output . . . 1070

L Node Execution Filters And Execution Multiplexers 1073
L.1 Data Producers: Default Configuration For Running And Sampling . . . 1074
L.1.1 Nodes That Data Producers Are Running On By Default—The `nodes` Command . . . 1074
L.1.2 Nodes That Data Producers Target By Default—The `samplenow` Command . . . 1074
L.2 Data Producers: Configuration For Running And Targeting . . . 1075
L.2.1 Custom Metrics From The `fm.sh` Custom Script . . . 1075
L.3 Replacing A Resource With An Explicit Node Specification . . . 1077
L.4 Excessive Sampling . . . 1078
L.5 Not Just For Nodes . . . 1079
L.6 Lua Node Execution Filters . . . 1079

M A Tree View Of `cmsh` 1083
M.1 Modes . . . 1083

N Redfish Management 1089
N.1 Redfish Overview . . . 1089
N.1.1 Redfish Features . . . 1089
N.1.2 Redfish In BCM . . . 1089
N.2 Redfish Commands . . . 1090
N.2.1 Redfish Status . . . 1090
N.2.2 Hardware Inventory Using Redfish . . . 1090
N.2.3 Power Control Using Redfish . . . 1090
N.2.4 Redfish Subscriptions: `redfishsubscriptions` . . . 1090
N.2.5 Redfish Event Testing: `redfishevent` . . . 1091

N.2.6 Redfish HTTP Commands 1091

Preface

Welcome to the *Administrator Manual* for the NVIDIA Base Command Manager 11 (BCM) environment.

0.1 Quickstart

For readers who want to get a cluster up and running as quickly as possible with NVIDIA Base Command Manager, there is a quickstart installation guide in Chapter 1 of the *Installation Manual*.

0.2 About This Manual

The rest of this manual is aimed at helping system administrators configure, understand, and manage a cluster running BCM so as to get the best out of it.

The *Administrator Manual* covers administration topics which are specific to the BCM environment. Readers should already be familiar with basic Linux system administration, which the manual does not generally cover. Aspects of system administration that require a more advanced understanding of Linux concepts for clusters are explained appropriately.

This manual is not intended for users interested only in interacting with the cluster to run compute jobs. The *User Manual* is intended to get such users up to speed with the user environment and workload management system.

0.3 About The Manuals In General

Regularly updated versions of the NVIDIA Base Command Manager 11 manuals are available on updated clusters by default at `/cm/shared/docs/cm`. The latest updates are always online at `https://docs.nvidia.com/base-command-manager`.

- The *Administrator Manual* describes the general administration of the cluster.
- The *Installation Manual* describes installation procedures.
- The *User Manual* describes the user environment and how to submit jobs for the end user.
- The *Cloudbursting Manual* describes how to deploy the cloud capabilities of the cluster.
- The *Developer Manual* has useful information for developers who would like to carry out programming tasks with BCM.
- The *Edge Manual* describes how to install and configure machine learning capabilities with BCM.
- The *Containerization Manual* describes how to manage containers with BCM.
- The *NVIDIA Mission Control Manual* describes NVIDIA Mission Control capabilities and integration with BCM.

If the manuals are downloaded and kept in one local directory, then in most pdf viewers, clicking on a cross-reference in one manual that refers to a section in another manual opens and displays that section in the second manual. Navigating back and forth between documents is usually possible with keystrokes or mouse clicks.

For example: `<Alt>-<Backarrow>` in Acrobat Reader, or clicking on the bottom leftmost navigation button of xpdf, both navigate back to the previous document.

The manuals constantly evolve to keep up with the development of the BCM environment and the addition of new hardware and/or applications. The manuals also regularly incorporate feedback from administrators and users, who can submit comments, suggestions or corrections via the website

`https://enterprise-support.nvidia.com/s/create-case`

Section 14.2 of the *Administration Manual* has more details on submitting an issue.

0.4 Getting Administrator-Level Support

Support for BCM subscriptions from version 10 onwards is available via the NVIDIA Enterprise Support page at:

`https://www.nvidia.com/en-us/support/enterprise/`

Section 14.2 has more details on working with support.

0.5 Getting Professional Services

The BCM support team normally differentiates between

- regular support (customer has a question or problem that requires an answer or resolution), and
- professional services (customer asks for the team to do something or asks the team to provide some service).

Professional services can be provided via the NVIDIA Enterprise Services page at:
`https://www.nvidia.com/en-us/support/enterprise/services/`

1

Introduction

1.1 NVIDIA Base Command Manager Functions And Aims

NVIDIA Base Command Manager (often shortened to BCM) contains tools and applications to facilitate the installation, administration, and monitoring of a cluster. In addition, BCM aims to provide users with an optimal environment for developing and running applications that require extensive computational resources.

1.2 The Scope Of The Administrator Manual (This Manual)

The *Administrator Manual* covers installation, configuration, management, and monitoring of BCM, along with relevant background information to help understand the topics covered.

1.2.1 Installation

Installation can generally be divided into parts as follows, with some parts covered by the *Administrator Manual*, some by the *Installation Manual*, and some by other manuals:

- **Initial installation of BCM:** This is covered in the *Installation Manual*, which gives a short introduction to the concept of a cluster along with details on installing BCM onto the head node. The *Installation Manual* is therefore the first manual an administrator should usually turn to when getting to work with BCM for the first time. The *Administrator Manual* can be referred to as the main reference resource once the head node has had BCM installed on it.

- **Provisioning installation:** This is covered in the *Administrator Manual*. After the head node has had BCM installed on it, the other, regular, nodes can (network) boot off it and provision themselves from it with a default image, without requiring a Linux distribution DVD themselves. The network boot and provisioning process for the regular nodes is described in detail in Chapter 5.

  In brief, provisioning installs an operating system and files on a node. This kind of installation to a regular node differs from a normal Linux installation in several ways. An important difference is that content that is put on the filesystem of the regular node is normally overwritten by provisioning when the regular node reboots.

- **Post-installation software installation:** The installation of software to a cluster that is already configured and running BCM is described in detail in Chapter 9 of this manual.

- **Third-party software installation:** The installation of software that is not developed as part of BCM, but is supported as a part of BCM. This is described in detail in the *Installation Manual*.

- **Cloudbursting, and Edge**: these are integrated as part of BCM in various ways. These have their own deployment procedures and have separate manuals.

1.2.2 Configuration, Management, And Monitoring Via BCM Tools And Applications

The administrator normally deals with the cluster software configuration via a front end to BCM. This can be GUI-based (Base View, section 2.4) or shell-based (`cmsh`, section 2.5). Other tasks can be handled via special tools provided with BCM, or the usual Linux tools. The use of BCM tools is usually recommended over standard Linux tools because cluster administration often has special issues, including that of scale.

The following topics are among those covered in this manual:

Chapter	Title	Description
2	Cluster Management With NVIDIA Base Command Manager	Introduction to main concepts and tools of BCM. Lays down groundwork for the remaining chapters
3	Configuring The Cluster	Further configuration and set up of the cluster after software installation of BCM on the head node.
4	Power Management	How power management within the cluster works
5	Node Provisioning	Node provisioning in detail
6	User Management	Account management for users and groups
7	Workload Management	Workload management implementation and use
8	The `cm-scale` Service	A BCM service to dynamically scale the cluster according to need
9	Post-Installation Software Management	Managing, updating, modifying BCM software and images
10	Monitoring: Monitoring Cluster Devices	Device monitoring and conditional action triggers
11	Monitoring: Job Monitoring	Jobs resource consumption monitoring by the jobs
12	Monitoring: Job Accounting	Jobs resource consumption monitoring aggregated by user or similar groupings
13	Monitoring: Job Chargeback	Resource request monitoring, so that groups of users can be charged for their use
14	Day-To-Day Administration	Miscellaneous administration
15	High Availability	Background details and setup instructions to build a cluster with redundant head nodes
16	The Jupyter Notebook Environment Integration	Installing and using the Jupyter notebook environment

The appendices to this manual generally give supplementary details to the main text.

The following topics are also logically a part of BCM administration, but they have their own separate manuals. This is because they have, or are eventually expected to have, many features or cover a special set of topics:

- Cloudbursting (*Cloudbursting Manual*)
- Edge deployment (*Edge Manual*)
- Developer topics (*Developer Manual*)
- Containerization topics (*Containerization Manual*)
- NVIDIA Mission Control topics (*NVIDIA Mission Control Manual*)

1.3 Outside The Direct Scope Of The Administrator Manual

The following supplementary resources can deal with issues related to this manual, but are outside its direct scope:

- **Use by the end user:** This is covered very peripherally in this manual. The user normally interacts with the cluster by logging into a custom Linux user environment to run jobs. Details on running jobs from the perspective of the user are given in the *User Manual*.
- **The NVIDIA Enterprise Support Portal** at `https://enterprise-support.nvidia.com/` carries Knowledge Articles that often supplement the *Administrator Manual* with discussion of the following:
  - Obscure, or complicated, configuration cases
  - Procedures that are not really within the scope of BCM itself, but that may come up as part of related general Linux configuration.
- **Further support options.** If the issue is not described adequately in the manuals, then section 14.2 describes how to get further support.

2

Cluster Management With NVIDIA Base Command Manager

This chapter introduces cluster management with NVIDIA Base Command Manager. A cluster running BCM exports a cluster management interface to the outside world, which can be used by any application designed to communicate with the cluster.

Section 2.1 introduces a number of concepts which are key to cluster management using BCM.

Section 2.2 gives a short introduction on how the modules environment can be used by administrators. The modules environment provides facilities to control aspects of a users' interactive sessions and also the environment used by compute jobs.

Section 2.3 introduces how authentication to the cluster management infrastructure works and how it is used. Section 2.4 and section 2.5 introduce the cluster management GUI (Base View) and cluster management shell (`cmsh`) respectively. These are the primary applications that interact with the cluster management daemon.

Section 2.6 describes the basics of the cluster management daemon, CMDaemon, running on all nodes of the cluster.

2.1 Concepts

In this section some concepts central to cluster management with BCM are introduced.

2.1.1 Devices

A *device* in BCM infrastructure represents components of a cluster. A device can be any of the following types:

- Head Node
- Physical Node
- Virtual Node
- Cloud Node
- GPU Unit
- Chassis
- Switch (ethernet, InfiniBand, Myrinet)
- Lite Node

- Power Distribution Unit
- Rack Sensor Kit
- Generic Device

A device can have a number of properties (e.g. rack position, hostname, switch port) which can be set in order to configure the device. Using BCM, operations (e.g. power on) may be performed on a device. The property changes and operations that can be performed on a device depend on the type of device. For example, it is possible to mount a new filesystem to a node, but not to an Ethernet switch.

Every device that is managed by BCM has a device state associated with it. The table below describes the most important states for devices:

device statuses	device is	monitored by BCM?	state tracking?
`[ UP ]`	UP	monitored	tracked
`[ DOWN ]`	DOWN	monitored	tracked
`[ CLOSED ] (UP)`	UP	mostly ignored	tracked
`[ CLOSED ] (DOWN)`	DOWN	mostly ignored	tracked

These, and other states are described in more detail in section 5.5.

`[ DOWN ]` and `[ CLOSED ] (DOWN)` states have an important difference. In the case of `[ DOWN ]`, the device is down, but is typically intended to be available, and thus typically indicates a failure. In the case of `[ CLOSED ] (DOWN)`, the device is down, but is intended to be unavailable, and typically indicates that the administrator deliberately brought the device down, and would like the device to be ignored.

2.1.2 Software Images

A *software image* is a blueprint for the contents of the local filesystems on a regular node. In practice, a software image is a directory on the head node containing a full Linux filesystem.

The software image in a standard BCM installation is based on the same parent distribution that the head node uses. A different distribution can also be chosen after installation, from the distributions listed in section 2.1 of the *Installation Manual* for the software image. That is, the head node and the regular nodes can run different parent distributions. However, such a "mixed" cluster can be harder to manage and it is easier for problems to arise in such mixtures. Such mixtures, while supported, are therefore not recommended, and should only be administered by system administrators that understand the differences between Linux distributions.

RHEL 8 and Rocky Linux 8 mixtures are completely compatible with each other on the head and regular nodes. The same applies to RHEL9 and Rocky Linux 9. That is because Rocky Linux is designed to be a binary-compatible derivative of its RHEL parents. On the other hand, SLES and Ubuntu need quite some effort to work in a mixture.

When a regular node boots, the node provisioning system (Chapter 5) sets up the node with a copy of the software image, which by default is called `default-image`.

Once the node is fully booted, it is possible to instruct the node to re-synchronize its local filesystems with the software image. This procedure can be used to distribute changes to the software image without rebooting nodes (section 5.6.2).

It is also possible to "lock" a software image so that no node is able to pick up the image until the software image is unlocked. (section 5.4.7).

Software images can be changed using regular Linux tools and commands (such as `rpm` and `chroot`). More details on making changes to software images and doing image package management can be found in Chapter 9.

2.1.3 Node Categories

Reasons For Categories

The collection of settings in BCM that can apply to a node is called the configuration of the node. The administrator usually configures nodes using the Base View (section 2.4) or `cmsh` (section 2.5) front end tools, and the configurations are managed internally with a database.

A *node category* is a group of regular nodes that share the same configuration. Node categories allow efficiency, allowing an administrator to:

- configure a large group of nodes at once. For example, to set up a group of nodes with a particular disk layout.
- operate on a large group of nodes at once. For example, to carry out a reboot on an entire category.

A regular node is in exactly one category at all times, which is `default` by default. The default category can be changed by accessing the `base` object of `partition` mode (page 113), and setting the value of `defaultcategory` to another, existing, category.

Nodes are typically divided into node categories based on the hardware specifications of a node or based on the task that a node is to perform. Whether or not a number of nodes should be placed in a separate category depends mainly on whether the configuration—for example: monitoring setup, disk layout, role assignment—for these nodes differs from the rest of the nodes.

Corresponding Category Values And Node Values

- For non-Boolean values, a node inherits values from the category it is in. Each value is treated as the default property value for a node, and can be overruled by specifying the node property value for a particular node.
- For Boolean values, such as `datanode` (page 280) and `installbootrecord` (page 287), a node does not inherit the value from the category it is in. Instead the category Boolean value has the Boolean `or` operation applied to the node Boolean value, and the result is the Boolean value that is used for the node. Since by default the values for these Booleans are zero, the result is initially similar to the non-Boolean values behavior.

Category And Software Image Do Not Necessarily Map One-To-One

One configuration property value of a node category is its software image (section 2.1.2). However, there is no requirement for a one-to-one correspondence between node categories and software images. Therefore multiple node categories may use the same software image, and conversely, one variable image—it is variable because it can be changed by the node setting—may be used in the same node category.

Software images can have their parameters overruled by the category settings. By default, however, the category settings that can overrule the software image parameters are unset.

By default, all nodes are placed in the `default` category. Alternative categories can be created and used at will, such as:

Example

Node Category	Description
`nodes-ib`	nodes with InfiniBand capabilities
`nodes-highmem`	nodes with extra memory
`login`	login nodes
`storage`	storage nodes

2.1.4 Node Groups

A *node group* consists of nodes that have been grouped together for convenience. The group can consist of any mix of all kinds of nodes, irrespective of whether they are head nodes or regular nodes, and irrespective of what category they are in. A node may be in 0 or more node groups at one time. I.e.: a node may belong to many node groups.

Node groups are used mainly for carrying out operations on an entire group of nodes at a time. Since the nodes inside a node group do not necessarily share the same configuration, configuration changes cannot be carried out using node groups.

Example

Node Group	Members
`brokenhardware`	`node087 node783 node917`
`headnodes`	`mycluster-m1 mycluster-m2`
`rack5`	`node212..node254`
`top`	`node084 node126 node168 node210`

One important use for node groups is in the `nodegroups` property of the provisioning role configuration (section 5.2.1), where a list of node groups that provisioning nodes provision is specified.

2.1.5 Roles

A *role* is a task that can be performed by a node. By assigning a certain role to a node, an administrator activates the functionality that the role represents on this node. For example, a node can be turned into provisioning node, or can be turned into a storage node, by assigning the corresponding roles to the node. Roles typically have services (section 3.15) that are associated with them. For example, the `dhcpd` service is associated with the `boot` role, which means that nodes assigned a `boot` role can then provide netbooting using `dhcpd` to other nodes.

Roles can be assigned to individual nodes or to node categories. When a role has been assigned to a node category, it is implicitly assigned to all nodes inside the category.

A *configuration overlay* (section 2.1.6) is a group of roles that can be assigned to designated groups of nodes within a cluster. This allows configuration of a large number of configuration parameters in various combinations of nodes.

Some roles allow parameters to be set that influence the behavior of the role. For example, the `Slurm Client Role` (which turns a node into a Slurm client) uses parameters to control how the node is configured within Slurm in terms of queues and the number of GPUs.

When a role has been assigned to a node category with a certain set of parameters, it is possible to override the parameters for a node inside the category. This can be done by assigning the role again to the individual node with a different set of parameters. Roles that have been assigned to nodes override roles that have been assigned to a node category.

Roles have a priority setting associated with them. Roles assigned at category level have a fixed priority of 250, while roles assigned at node level have a fixed priority of 750. The configuration overlay priority is variable, but is set to 500 by default. Thus, for example, roles assigned at the node level override roles assigned at the category level. Roles assigned at the node level also override roles assigned by the default configuration overlay.

A role can be imported from another entity, such as a node, a category, or a configuration overlay.

Roles are assigned only from existing roles. A custom role cannot be created by the cluster administrator, which means that the `add` command (page 57) does not work in `role` mode, by design. Instead, only the `assign` command (page 67) can be used for roles.

Examples of role assignment are given in sections 5.2.2 and 5.2.3.

2.1.6 Configuration Overlay

A configuration overlay assigns roles (section 2.1.5) for groups of nodes. The number of roles can be quite large, and priorities can be set for these.

Multiple configuration overlays can be set for a node. A priority can be set for each configuration overlay, so that a configuration overlay with a higher priority is applied to its associated node instead of a configuration overlay with a lower priority. The configuration overlay with the highest priority then determines the actual assigned role.

A configuration overlay assigns a group of roles to an instance. This means that roles are assigned to nodes according to the instance configuration, along with a priority. Whether the configuration overlay assignment is used, or whether the original role assignment is used, depends upon the configured priorities.

Configuration overlays can take on priorities in the range 0-1000, except for 250 and 750, which are forbidden. Setting a priority of -1 means that the configuration overlay is ignored.

The priorities of 250, 500, and 750 are also special, as indicated by the following table:

priority	assigned to node from
-1	*configuration overlay not assigned*
250	category
500	configuration overlay with default priority
750	node

A custom configuration overlay can be created by the cluster administrator in `configurationoverlay` mode, which means that the `add` command (page 57) does work for it. This can be used to achieve a functional equivalent to creating custom roles. The creation of custom roles in `role` mode is a function that is missing by design.

2.2 Modules Environment

The *modules environment* is the shell environment that is set up by a third-party software (section 7.1 of the *Installation Manual*) called Environment Modules. The software allows users to modify their shell environment using pre-defined *modules*. A module may, for example, configure the user's shell to run a certain version of an application.

Details of the modules environment from a user perspective are discussed in section 2.3 of the *User Manual*. However some aspects of it are relevant for administrators and are therefore discussed here.

2.2.1 Adding And Removing Modules

Modules may be loaded and unloaded, and also be combined for greater flexibility.

Modules currently installed are listed with:

```
module list
```

The modules available for loading are listed with:

```
module avail
```

Loading and removing specific modules is done with `module load` and `module remove`, using this format:

```
module load <module name 1> [<module name 2> ...]
```

For example, loading the `shared` module (section 2.2.2), the `gcc` compiler, the `openmpi` parallel library, and the `openblas` library, allows an MPI application `myapp.c` to be compiled with OpenBLAS optimizations:

Example

```
module add shared
module add gcc/13.1.0
module add openmpi/gcc/64/4.1.5
module add openblas
module add openblas/dynamic/0.3.18
mpicc -o myapp myapp.c
```

The exact versions used can be selected using tab-completion. In most cases, specifying version numbers explicitly is typically only necessary when multiple versions of an application are installed and available. When there is no ambiguity, module names without a further path specification may be used.

2.2.2 Using Local And Shared Modules

Applications and their associated modules are divided into *local* and *shared* groups. Local applications are installed on the local filesystem, whereas shared applications reside on a shared (i.e. imported) filesystem.

It is recommended that the `shared` module be loaded by default for ordinary users. Loading it gives access to the modules belonging to shared applications, and allows the `module avail` command to show these extra modules.

Loading the `shared` module automatically for `root` is not recommended on a cluster where shared storage is not on the head node itself. This is because `root` logins could be obstructed if this storage is not available, and if the `root` user relies on files in the shared storage.

On clusters without external shared storage, `root` can safely load the `shared` module automatically at login. This can be done by running the following command as `root`:

```
module initadd shared
```

Other modules can also be set to load automatically by the user at login by using "`module initadd`" with the full path specification. With the `initadd` option, individual users can customize their own default modules environment.

Modules can be combined in *meta-modules*. By default, the `default-environment` meta-module exists, which allows the loading of several modules at once by a user. Cluster administrators are encouraged to customize the `default-environment` meta-module to set up a recommended environment for their users. The `default-environment` meta-module is empty by default.

The administrator and users have the flexibility of deciding the modules that should be loaded in undecided cases via module *dependencies*. Dependencies can be defined using the `prereq` and `conflict` commands. The man page for `modulefile` gives details on configuring the loading of modules with these commands.

2.2.3 Setting Up A Default Environment For All Users

How users can set up particular modules to load automatically for their own use with the `module initadd` command is discussed in section 2.2.2.

How the administrator can set up particular modules to load automatically for all users by default is discussed in this section (section 2.2.3). In this example it is assumed that all users have just the following modules as a default:

Example

```
[fred@basecm11 ~]$ module list
Currently Loaded Modulefiles:
1) shared
```

The slurm and gdb modules can then be set up by the administrator as a default for all users in the following 2 ways:

1. Creating and defining part of a .profile to be executed for login shells. For example, a file userdefaultmodules.sh created by the administrator:

```
[root@basecm11 ~]# cat /etc/profile.d/userdefaultmodules.sh
module load shared
module load slurm
module load gdb
```

   Whenever users now carry out a bash login, these modules are loaded.

2. Instead of placing the modules directly in a script under profile.d like in the preceding item, a slightly more sophisticated way is to set the modules in the meta-module /cm/shared/modulefiles/default-environment. For example:

```
[root@basecm11 ~]# cat /cm/shared/modulefiles/default-environment
#%Module1.0#####################################################
## default modulefile
##
proc ModulesHelp { } {
        puts stderr "\tLoads default environment modules for this cluster"
}
module-whatis   "adds default environment modules"

# Add any modules here that should be added by when a user loads the 'default-enviro\
nment' module
module add shared slurm gdb
```

   The script userdefaultmodules.sh script under profile.d then only needs to have the default-environment module loaded in it:

```
[root@basecm11 ~]# cat /etc/profile.d/userdefaultmodules.sh
module load -s default-environment
```

   The -s option is used to load it silently, because otherwise a message is displayed on the terminal informing the person logging in that the default-environment module has been loaded.

   Now, whenever the administrator changes the default-environment module, users get these changes too during login.

The lexicographical order of the scripts in the /etc/profile directory is important. For example, naming the file defaultusermodules.sh instead of userdefaultmodules.sh means that the modules.sh script is run after the file is run, instead of before, which would cause an error.

2.2.4 Creating A Modules Environment Module

All module files are located in the /cm/local/modulefiles and /cm/shared/modulefiles directories. A module file is a Tcl or Lua script in which special commands are used to define functionality. The modulefile(1) man page has more on this.

Cluster administrators can use the existing modules files as a guide to creating and installing their own modules for module environments, and can copy and modify a file for their own software if there is no environment provided for it already by BCM.

2.2.5 Lua Modules Environment (Lmod)

By default, BCM uses traditional Tcl scripts for its environment modules files, or *Tmod* in BCM shorthand. Lua modules, or *Lmod*, provide alternative environment modules files, where the files are typically written in Lua. Lmod can be used as a replacement for Tmod.

Conceptually Lmod works in the same way as Tmod, but provides some extra features and commands.

For Lmod, the module files are typically written in Lua, but Lmod is also capable of reading Tcl module files. It is therefore not necessary to convert all existing Tcl modules manually to the Lua language.

On a BCM cluster, both Lmod and Tmod are installed by default. However only one of them is active, depending on which one is enabled. Switching between Lmod and Tmod for a node can be done by setting an environment variable, `$ENABLE_LMOD` in the `cm-lmod-init.sh` shell script.

Switching For The Head Node

For example, for the head node:

Example

```
[root@basecm11 ~]# cat /etc/sysconfig/modules/lmod/cm-lmod-init.sh
export ENABLE_LMOD=1
```

In the preceding example, Lmod is enabled, and Tmod is disabled because `$ENABLE_LMOD` is set to `1`.

Example

```
[root@basecm11 ~]# cat /etc/sysconfig/modules/lmod/cm-lmod-init.sh
export ENABLE_LMOD=0
```

In the preceding example, Lmod is disabled, and Tmod is enabled because `$ENABLE_LMOD` is set to `0`.

A change in the file on the node is effective after having logged out, then logged into the shell again.

Switching For The Regular Nodes

A *node image* is a directory and contents of that directory. It is used as the template for a regular node when the node is provisioned (Chapter 5). For a node image with the name <*image name*>, the `cm-lmod-init.sh` file is located at `/cm/images/`<*image name*>`/etc/sysconfig/modules/lmod/cm-lmod-init.sh`. For switching between Lmod and Tmod on a regular node, the file is changed on the image, and the file on the image is then updated to the node. The update from the image to the node is typically carried out with the `imageupdate` command in `cmsh` (section 5.6.2) or the `Update node` command in Base View (section 5.6.3).

2.3 Authentication

2.3.1 Changing Administrative Passwords On The Cluster

How to set up or change regular user passwords is not discussed here, but in Chapter 6 on user management.

Amongst the administrative passwords associated with the cluster are:

1. **The root password of the head node:** This allows a root login to the head node.
2. **The root passwords of the software images:** These allow a root login to a regular node running with that image, and is stored in the image file.
3. **The root password of the node-installer:** This allows a root login to the node when the node-installer, a stripped-down operating system, is running. The node-installer stage prepares the node for the final operating system when the node is booting up. Section 5.4 discusses the node-installer in more detail.

4. **The root password of MySQL:** This allows a root login to the MySQL server.

To avoid having to remember the disparate ways in which to change these 4 kinds of passwords, the `cm-change-passwd` command runs a dialog prompting the administrator on which of them, if any, should be changed, as in the following example:

```
[root@basecm11 ~]# cm-change-passwd
With this utility you can easily change the following passwords:
  * root password of head node
  * root password of slave images
  * root password of node-installer
  * root password of mysql

Note: if this cluster has a high-availability setup with 2 head
      nodes, be sure to run this script on both head nodes.

Change password for root on head node? [y/N]: y
Changing password for root on head node.
Changing password for user root.
New password:
Retype new password:
passwd: all authentication tokens updated successfully.

Change password for root in default-image [y/N]: y
Changing password for root in default-image.
Changing password for user root.
New password:
Retype new password:
passwd: all authentication tokens updated successfully.

Change password for root in node-installer? [y/N]: y
Changing password for root in node-installer.
Changing password for user root.
New password:
Retype new password:
passwd: all authentication tokens updated successfully.

Change password for MYSQL root user? [y/N]: y
Changing password for MYSQL root user.
Old password:
New password:
Re-enter new password:
```

For a high-availability—also called a failover—configuration, the passwords are copied over automatically to the other head node when a change is made in the software image root password (case 2 on page 34).

For the remaining password cases (head root password, MySQL root password, and node-installer root password), the passwords are best "copied" over to the other head node by simply rerunning the script on the other head node.

Also, in the case of the password for software images used by the regular nodes: the new password that is set for a regular node only works on the node after the image on the node itself has been updated, with, for example, the `imageupdate` command (section 5.6.2). Alternatively, the new password can be made to work on the node by simply rebooting the node to pick up the new image.

The LDAP root password is a random string set during installation. Changing this is not done using `cm-change-password`. It can be changed as explained in Appendix I.

If the administrator has stored the password to the cluster in the Base View front-end, then the password should be modified there too (figure 2.2).

2.3.2 Logins Using `ssh`

The standard system login root password of the head node, the software image, and the node-installer, can be set using the `cm-change-passwd` command (section 2.3.1).

In contrast, `ssh` logins from the head node to the regular nodes are set by default to be passwordless:

- For non-root users, an `ssh` passwordless login works if the `/home` directory that contains the authorized keys for these users is mounted. The `/home` directory is mounted by default on the head node as well as on the regular node, so that by default a passwordless login works from the head node to the regular nodes, as well as from the regular nodes to the head node.
- For the root user, an `ssh` passwordless login should always work from the head node to the regular nodes since the authorized keys are stored in `/root`. Logins from the regular node to the head node are configured by default to request a password, as a security consideration.

Users can be restricted from `ssh` logins

- on regular nodes using the `cmsh` `usernodelogin` option (section 7.2.1) or the Base View `User node login` (section 7.2.2) settings
- on the head node by modifying the sshd configuration on the head node. For example, to allow only root logins, the value of `AllowUsers` can be set in `/etc/ssh/sshd_config` to `root`. The man page for `sshd_config` has details on this.

2.3.3 Certificates

PEM Certificates And CMDaemon Front-end Authentication

While nodes in the cluster accept ordinary `ssh`-based logins, the cluster manager accepts public key authentication using `X509v3` certificates. Public key authentication using `X509v3` certificates means in practice that the person authenticating to the cluster manager must present their public certificate, and in addition must have access to the private key that corresponds to the certificate.

BCM uses the PEM format for certificates. In this format, the certificate and private key are stored as plain text in two separate PEM-encoded files, ending in `.pem` and `.key`.

Using `cmsh` and authenticating to BCM: By default, one administrator certificate is created for root for the `cmsh` front end to interact with the cluster manager. The certificate and corresponding private key are thus found on a newly-installed BCM cluster on the head node at:

```
/root/.cm/admin.pem
/root/.cm/admin.key
```

The `cmsh` front end, when accessing the certificate and key pair as user `root`, uses this pair by default, so that prompting for authentication is then not a security requirement. The logic that is followed to access the certificate and key by default is explained in detail in item 2 on page 346.

Using Base View and authenticating to BCM: When an administrator uses the Base View front end, a login to the cluster is carried out with username password authentication (figure 2.2), unless the authentication has already been stored in the browser, or unless certificate-based authentication is used.

- Certificate-based authentication can be carried out using a PKCS#12 certificate file. This can be generated from the PEM format certificates. For example, for the root user, an `openssl` command that can be used to generate the `admin.pfx` file is:

```
openssl pkcs12 -export -in ~/.cm/admin.pem -inkey ~/.cm/admin.key -out ~/.cm/admin.pfx
```

- In Chrome, the IMPORT wizard at chrome://settings/certificates can be used to save the file into the browser.
- For Firefox, the equivalent navigation path is:
  about:preferences#privacy > Certificates > View Certificates > Your Certificates > Import

The browser can then access the Base View front end without a username/password combination.

If the administrator certificate and key are replaced, then any other certificates signed by the original administrator certificate must be generated again using the replacement, because otherwise they will no longer function.

Certificate generation in general, including the generation and use of non-administrator certificates, is described in greater detail in section 6.5.

Replacing A Temporary Or Evaluation License

In the preceding section, if a license is replaced, then regular user certificates need to be generated again. Similarly, if a temporary or evaluation license is replaced, regular user certificates need to be generated again. This is because the old user certificates are signed by a key that is no longer valid. The generation of non-administrator certificates and how they function is described in section 6.5.

Checking Certificates Validity With cm-check-certificates.sh

A BCM script that checks whether certificates are current or expired is /cm/local/apps/cmd/scripts/cm-check-certificates.sh:

Example

```
root@basecm11:~# /cm/local/apps/cmd/scripts/cm-check-certificates.sh
/cm/local/apps/cmd/etc/cluster.pem: OK
/cm/local/apps/cmd/etc/cert.pem: OK

/cm/local/apps/cmd/etc/cluster.key : matches

All /cm/local/apps/cmd/etc/cluster.pem files are up to date (82070dedb489df6c19ffa3ace1bf354e)

/root/.cm/admin.pem: OK
 ... output truncated ...
root@basecm11:~#
```

2.3.4 Profiles

Certificates that authenticate to CMDaemon contain a *profile*.

A profile determines which cluster management operations the certificate holder may perform. The administrator certificate is created with the admin profile, which is a built-in profile that allows all cluster management operations to be performed. In this sense it is similar to the root account on unix systems. Other certificates may be created with different profiles giving certificate owners access to a pre-defined subset of the cluster management functionality (section 6.5).

2.4 Base View GUI

This section introduces the basics of the cluster management GUI (Base View). Base View is the web application front end to cluster management in BCM.

Base View is supported to run on the last 2 versions of Firefox, Google Chrome, Edge, and Safari. "Last 2 versions" means the last two publicly released versions (`https://browserl.ist/?q=last+2+versions`) at the time of release of NVIDIA Base Command Manager. For example, at the time of writing of this section, July 2026, the last 2 versions were:

Browser	Versions
Chrome	149, 150
Edge	149, 150
Firefox	151, 152
Safari	26.3, 26.4

Base View should run on more up-to-date versions of the browsers in the table without issues.

Base View should run on other recent browsers without issues too, but this is not supported. Browsers that run on mobile devices are also not supported.

2.4.1 Installing The Cluster Management GUI Service

In a default installation, accessing the head node hostname or IP address with a browser leads to the landing page (figure 2.1).

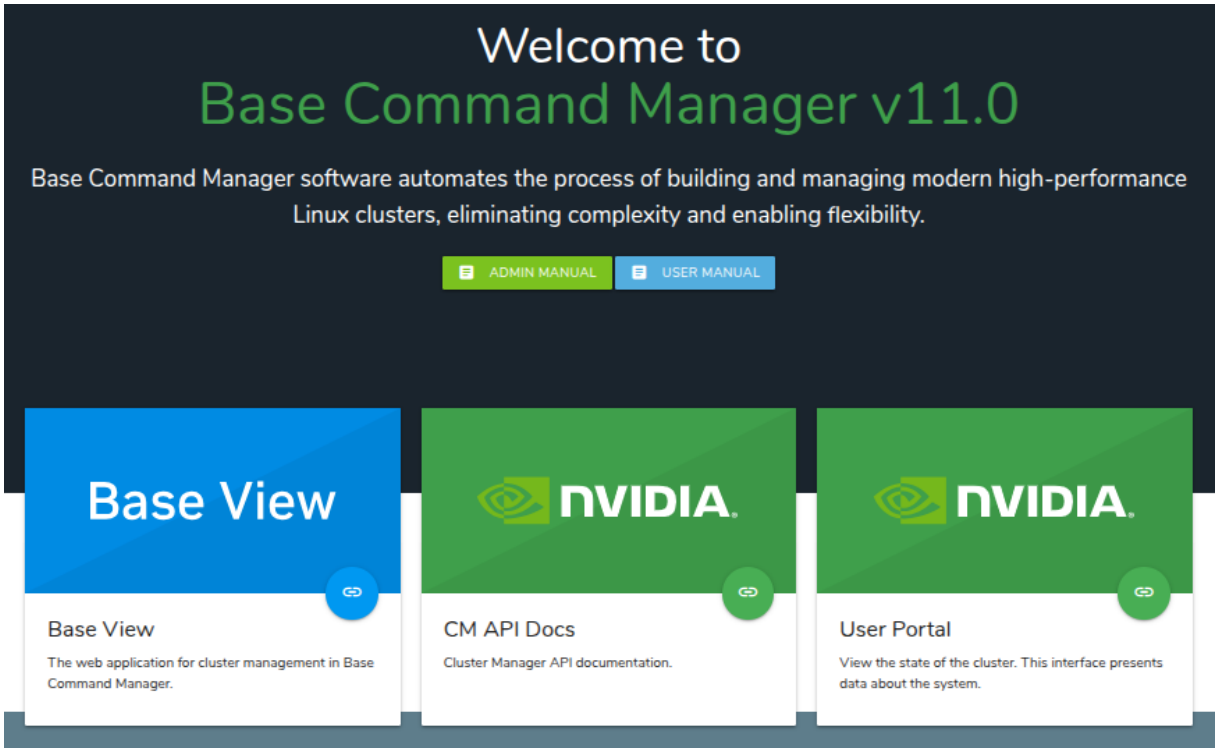


Figure 2.1: Head node hostname or IP address landing page at `https://`*<host name or IP address>*

The landing page is served by the Apache web server from the distribution, and can be served over the HTTP (unencrypted) or HTTPS (encrypted) protocols.

The certificates used to ensure an encrypted connection are set within:

- `/etc/httpd/conf.d/ssl.conf` for the RHEL family of distributions. The PEM-encoded certificate at `/etc/pki/tls/certs/localhost.crt` is set by default.
- `/etc/apache2/sites-available/default-ssl.conf` for Ubuntu. The PEM-encoded certificate at `/etc/ssl/certs/ssl-cert-snakeoil.pem` is set by default.

The system administrator may wish to consider the security aspects of using the default distribution certificates, and may wish to replace them.

Within the landing page are several blocks, one of which is the Base View block. Base View is the BCM GUI. Within the Base View block is a clickable link, which is a circle with a chain-link symbol inside it.

Base View connects by default to the encrypted web service on port 8081. This is served from the head node cluster manager, rather than from Apache, to the browser. The direct URL for this is of the form:

`https://`*<host name or IP address>*`:8081/base-view`

The BCM package that provides the service is `base-view` and it is installed by default with BCM. The service can be disabled by removing the package with, for example, `yum remove base-view`.

NVIDIA Base Command Manager Base View Login Window

Figure 2.2 shows the login dialog window for Base View.

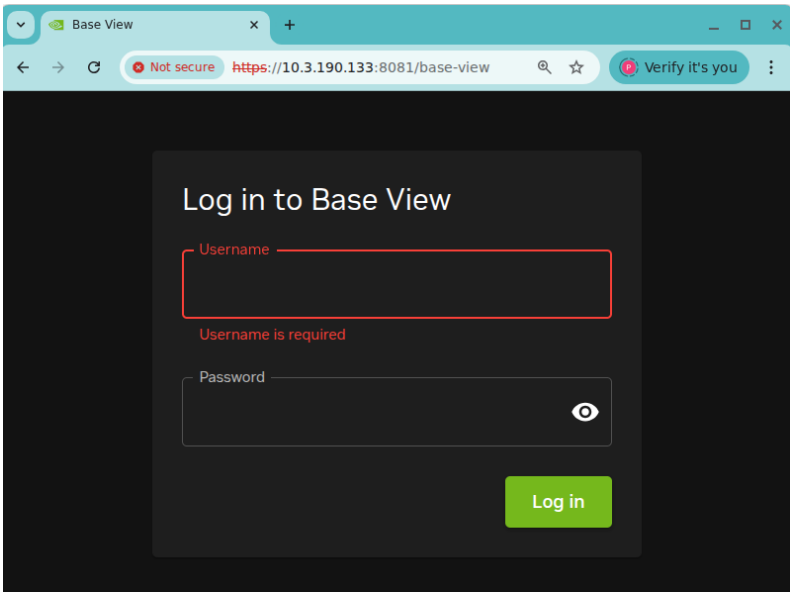


Figure 2.2: Base View Login via `https://`*<host name or IP address>*`:8081/base-view`

NVIDIA Base Command Manager `Base View` Default Display On Connection

Clicking on the `Login` button logs the administrator into the Base View service on the cluster. By default an overview window is displayed, corresponding to the navigation path `Cluster` > `Overview` (figure 2.3).

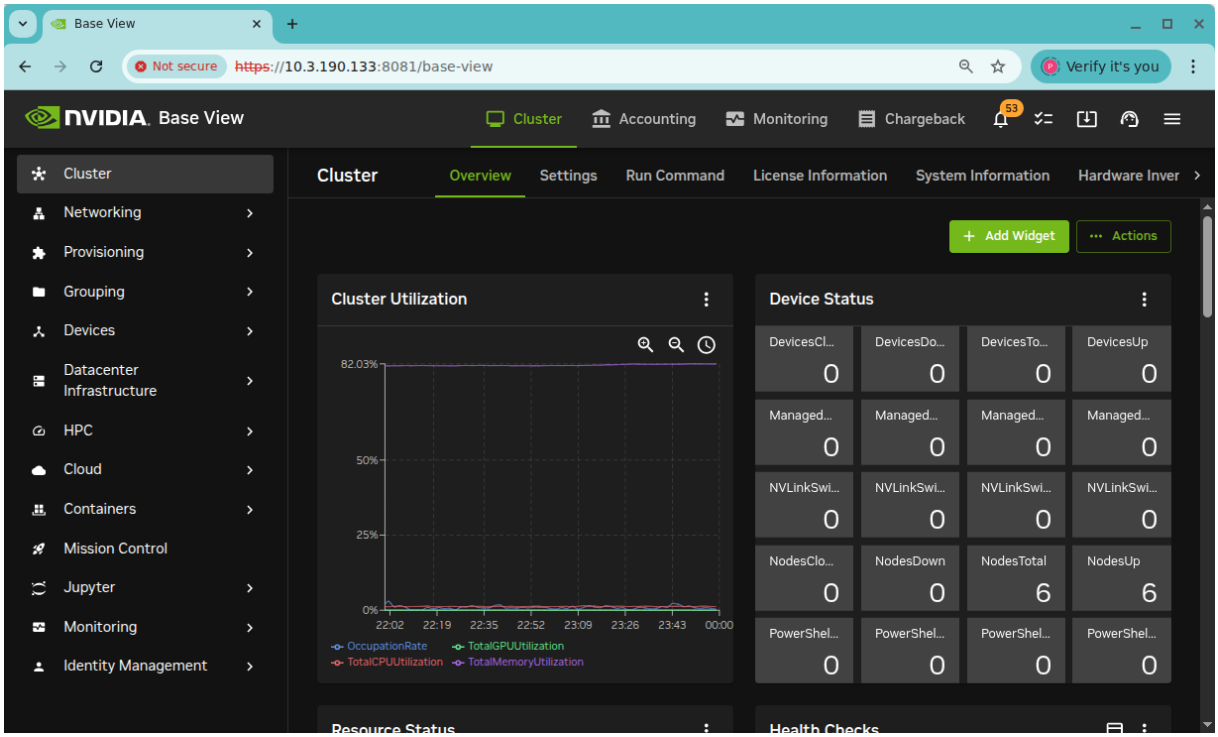


Figure 2.3: Cluster Overview

2.4.2 Navigating The Cluster With Base View

Aspects of the cluster can be managed by administrators using Base View (figure 2.3).

Navigating The Cluster With Base View: Getting Around

The resource tree, displayed on the left side of the window, consists of representations of cluster usage resources that are viewable or configurable, such as `Networking`, `Provisioning`, `Grouping`, and `Devices`, `Cloud`, and `Containers`.

Selecting a resource opens a window that allows parameters related to the resource to be viewed and managed.

As an example, the `Cluster` resource can be selected. On the first session this opens up the `Overview` window, which allows the cluster administrator to see the main characteristics of the cluster quickly. The characteristics can be explored further by clicking through on widgets that open up new windows, or clicking on tabs. Clicking on the BCM NVIDIA logo in the top left corner of the page conveniently brings the user back to `Cluster > Overview`.

Within the main content window, there are a variety of widgets that are used. A helpful nuance of widget behavior to note is that

- the ⧉ widget is for opening up an existing object, and
- the + widget is for opening up a new object.

Both widgets open up a subwindow in which values can be viewed, and perhaps modified.

Also useful to know is the following: going back to a previous window is possible by scrolling back up to the top of the subwindow and clicking on the appropriate item in the breadcrumbs trail. For example: A user has navigated down the path: `Cluster > Settings > Time zone >` ⧉ , and has reached a configuration screen to set the time zone, as in figure 2.4:

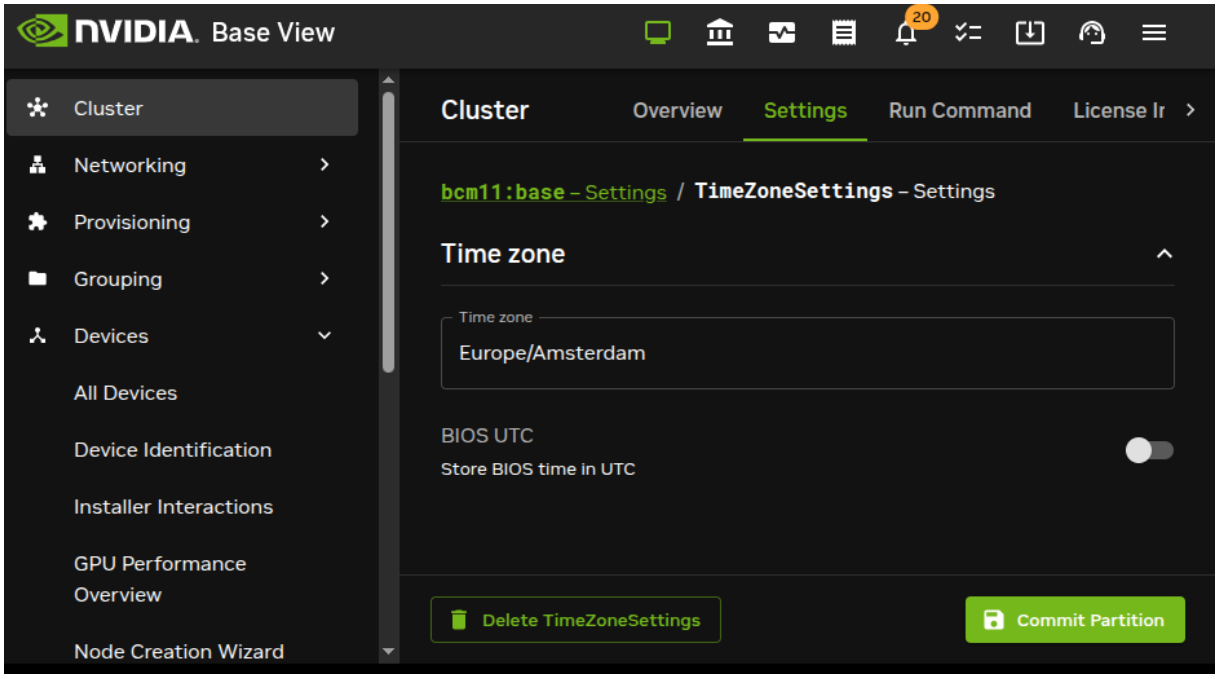


Figure 2.4: Base View Time Zone Breadcrumb

If the user wishes to return to the earlier `Cluster > Settings` screen without changing any values in the current time zone screen, then it can be done by clicking on the appropriate breadcrumb. In this case it is the breadcrumb associated with the navigation path to that earlier screen, which in figure 2.4 is `bcm11:base - Settings`.

Navigating The Cluster With Base View: A Tree Map Of Locations From The `Cluster` Resource

The tabs within the `Cluster` resource window are mapped out in the tree figure displayed next, and the tree is followed by a brief description of its items:

```
Cluster
|
|-- Overview
|      |
|      |-- Cluster Utilization
|      |      `-- displays a chart of cluster utilization against time for:
|      |          OccupationRate
|      |          TotalGPUUtilization
|      |          TotalCPUUtilization
|      |          TotalMemoryUtilization
|      |
|      |-- Job Slots Utilization
|      |      `-- displays charts of any job slots utilization
|      |
|      |-- Device Status
|      |      |-- Devices{Closed|Down|Total|Up}
|      |      |-- ManagedSwitches{Closed|Down|Total|Up}
|      |      |-- NVLinkSwitches{Closed|Down|Total|Up}
|      |      |-- Nodes{Closed|Down|Total|Up}
|      |      `-- PowerShelves{Closed|Down|Total|Up}
|      |
|      |-- Resource Status
|      |      |-- Cores{Down|Total|Up}
```

```
|     |     |-- DPUNodes{Down|Total|Up}
|     |     `-- GPUs{Down|Total|Up}
|     |
|     |-- Health Checks
|     |     |-- Entity
|     |     |-- Name
|     |     |-- Value
|     |     `-- Last Check
|     |
|     |-- GPUs
|     |     |-- Node
|     |     |-- GPU
|     |     |-- Utilization
|     |     |-- Temperature
|     |     `-- Power Usage
|     |
|     |-- Workload
|     |     |-- Name
|     |     |-- Scheduler
|     |     |-- Running
|     |     `-- Queued
|     |
|     |-- Cluster Overview
|     |     |-- Uptime
|     |     `-- Phase load
|     |
|     |-- Disks
|     |     |-- Mountpoint
|     |     |-- Used
|     |     `-- Free
|     |
|     `-- License Information
|           |-- Nodes
|           |-- Accelerators
|           |-- Building Management System Integration
|           |-- Leak Detection
|           |-- NMX Manager
|           |-- Inter-node Memory Exchange (IMEX) Management
|           |-- GB200 Rack Management
|           |-- GB200 Firmware Management
|           |-- Power Reservation Steering
|           |-- Run:ai
|           |-- GPU Workload Power Profiles
|           |-- Autonomous Hardware Recovery
|           `-- Autonomous Job Recovery
|
|-- Settings
|     |-- Cluster name
|     |     |-- Cluster name
|     |     |-- Cluster reference architecture
|     |     |-- Administrator email
|     |     |-- Name
|     |     `-- Headnode
|     |
|     |-- Name servers
```

```
|     |     |-- Name servers
|     |     |-- Name servers from DHCP
|     |     `-- Time servers
|     |
|     |-- Search domains
|     |     |-- Search domains
|     |     `-- Relay Host
|     |
|     |-- Externally visible IP
|     |     `-- Externally visible IP
|     |
|     |-- Time zone
|     |     `-- Time zone
|     |
|     |-- BMC settings
|     |     |-- BMC Settings
|     |     |-- SNMP Settings
|     |     |-- DPU Settings
|     |     |-- SELinux Settings
|     |     |-- Access Settings
|     |     |-- Provisioning Settings
|     |     |-- ZTP settings
|     |     |-- ZTP new switch settings
|     |     |-- NetQ settings
|     |     |-- UFM settings
|     |     `-- NMX Manager settings
|     |
|     |-- Default burn configuration
|     |     |-- Default burn config
|     |     `-- Available Burn configs
|     |
|     |-- External Network
|     |     |-- External network
|     |     |-- Management network
|     |     `-- No zero conf
|     |
|     |-- Default category
|     |     |-- Default category
|     |     `-- ArchOS
|     |
|     |-- Sign installer certificates
|     |     `-- Sign installer certificates
|     |
|     |-- Failover
|     |     `-- Failover
|     |
|     |-- Failover groups
|     |     `-- Failover groups
|     |
|     |-- Notes
|     |     `-- Notes
|     |
|     |-- Wlm job power usage settings
|     |     `-- Wlm job power usage settings
|     |
```

```
|     |-- Leak action policies
|     |     |-- Active leak action policies
|     |     `-- Leak action policies
|     |
|     |-- Building Management System
|     |     |-- BMS
|     |     |-- BMS path
|     |     |-- BMS Certificate
|     |     `-- BMS private key
|     |
|     `-- Prometheus metric forwarders
|           `-- Prometheus metric forwarders
|
|-- Run Command
|     |-- Nodes
|     `-- Command
|
|-- License Information
|     |-- License Information
|     |     |-- Edition
|     |     |-- Duration
|     |     |-- Version
|     |     |-- MAC Address
|     |     |-- Nodes
|     |     |-- Accelerators
|     |     |-- Licensee
|     |     `-- Serial
|     |
|     `-- Mission Control
|           |-- Manage Mission Control (hyperlink)
|           |-- Building Management System Integration
|           |-- Leak Detection
|           |-- NMX Manager
|           |-- Inter-node Memory Exchange (IMEX) Management
|           |-- GB200 Rack Management
|           |-- GB200 Firmware Management
|           |-- Power Reservation Steering
|           |-- Run:ai
|           |-- Autonomous Hardware Recovery
|           `-- Autonomous Job Recovery
|
|-- System Information
|     |-- Type
|     |-- Memory Total
|     |-- Disk count
|     |-- Hostname
|     |-- BIOS Date
|     |-- OS Version
|     |-- System Name
|     |-- Number of CPUs
|     `-- Number of Cores
|
|-- Hardware Inventory
|     |-- Device
|     |-- Name
```

```
|     |-- Manufacturer
|     |-- Model
|     |-- Serial Number
|     `-- Part Number
|
|-- Node Arch OS Information
|     |-- Hostname
|     |-- Reported Arch
|     |-- Reported OS
|     |-- Reported Timestamp
|     |-- Configured Arch
|     `-- Configured OS
|
|-- Version Information
|     |-- Hostname
|     |-- BCM Version
|     |-- CMDaemon Version
|     |-- Build Index
|     |-- Build Hash
|     `-- Database Version
|
|-- Port Forwarding
|     |-- Device
|     |-- IP
|     |-- Port
|     |-- Head Port
|     |-- Sessions
|     `-- Actions
|
|-- IMEX Configuration
|     |-- Node
|     |-- IP
|     |-- Status
|     |-- Version
|     |-- Connection Node
|     |-- Connection IP
|     `-- Connection Status
|
`-- Workload Utilization
      `-- displays a chart of jobs against nodes
```

Overview

The `Overview` tab window in the tree map has sections that show:

- `Cluster Utilization` graphs, which includes `Occupation rate` (page 995).
- The number of devices in various states (`UP`, `DOWN` ...) and their total number. Devices states are covered in sections 5.5.3 and 5.5.4.
- The number of resources (Cores, DPUs, and GPUs) in various states (`UP`, `DOWN`, ...) and their total number.
- Health checks (sections 10.2.1 and 10.2.4).
- GPU usage information (includes temperature, power).
- Workload scheduling. Workload management is described in Chapter 7.

- Uptime.
- Disks status.
- The license information. This shows the number of nodes and accelerators (such as GPUs) that have been used, out of the number that are licensed to be used. In addition, the license information section shows whether particular license features are enabled, such as for features available in NVIDIA Mission Control (section 1.2 of the *NVIDIA Mission Control Manual*).

Settings

The `Settings` window has a number of global cluster properties and property groups. These are loosely grouped as follows:

- `Cluster name`: This section can also manage the cluster administrator email, partition name, and head node name.
- `Name servers`: This section also manages the time servers.
- `Search domains`: A relay host can also be set in this section.
- `Externally visible IP`
- `Time zone`
- `BMC settings`: This section is also used to manage SNMP, DPU, SELinux, Access, Provisioning, ZTP, NetQ, UFM, and NMX Manager settings.
- `Default burn configuration`: Sets the default burn configuration, and values in the burn configuration settings.
- `External network`: The management network can be defined from this section.
- `Default category`: Sets the default category
- `Sign installer certificates`: This section manages the manual or automatic signing of the node-installer certificates (section 5.4.1).
- `Failover`: Opens up a window to manage failover properties.
- `Failover groups`: Opens up a window to manage failover groups properties.
- `Notes`
- `Wlm job power usage settings`: Opens up a new window to add settings
- `Leak action policies`: Opens up a new window to manage settings.
- `Building Management System`
- `Prometheus metric forwarders`: Opens up a new window to manage settings.

Run Command

The `Run command` tab window allows a specified command to be run on a selected node of the cluster.

License Information

The `License Information` section of the `License Information` tab window shows some further information to do with cluster licensing; it is the Base View version of `verify-license info` (section 4.2.2 of the *Installation Manual*).

A slightly obscure property within this pane is `Version`, which refers to the version type of the license, rather than the BCM version. The license for NVIDIA Base Command Manager, with a value for version of `10.0 and above`, is of a type that is incompatible with versions before 10.0. This means that an upgrade from before 9.2, to 10.0 or beyond, requires a license upgrade. The BCM support team must be contacted to arrange the license upgrade.

The `Mission Control` section of the `License Information` window lets NVIDIA Mission Control features be managed with the `Manage Mission Control` URL. Other parts of the section show which features of the NVIDIA Mission Control license are currently active.

System information

The `System information` tab window shows the main hardware specifications of the devices in the cluster (CPU, memory, BIOS), along with the operating system version that each device runs.

Hardware Inventory

The `Hardware Inventory` tab window displays a table of hardware identifiers. It is the Base View version of the `hardwareinventoryinfo` command in the `device` mode of `cmsh`.

Version Information

The `Version Information` tab window shows version information for important cluster software components, such as the CMDaemon database version, BCM version and builds.

Port Forwarding

The `Port Forwarding` tab window shows ports that have been forwarded from the head node to the regular nodes. It is the Base View version of the `portforward list` command (section 3.2.3) in the `device` mode of `cmsh`.

IMEX Configuration

The `IMEX configurations` tab window lists the state of IMEX mappings per node.

Workload Utilization

The `Workload Utilization` tab window displays a chart of nodes running a job or not.

2.5 Cluster Management Shell

This section introduces the basics of the cluster management shell, `cmsh`. This is the command-line interface to cluster management in BCM. Since `cmsh` and Base View give access to the same cluster management functionality, an administrator need not become familiar with both interfaces. Administrators intending to manage a cluster with only Base View may therefore safely skip this section.

The `cmsh` front end allows commands to be run with it, and can be used in batch mode. Although `cmsh` commands often use constructs familiar to programmers, it is designed mainly for managing the cluster efficiently rather than for trying to be a good or complete programming language. For programming cluster management, the use of Python bindings (Chapter 1 of the *Developer Manual*) is generally recommended instead of using `cmsh` in batch mode.

Usually `cmsh` is invoked from an interactive session (e.g. through `ssh`) on the head node, but it can also be used to manage the cluster from outside.

2.5.1 Invoking `cmsh`

From the head node, `cmsh` can be invoked as follows:

Example

```
[root@mycluster ~]# cmsh
[mycluster]%
```

By default it connects to the IP address of the local management network interface, using the default BCM port. If it fails to connect as in the preceding example, but a connection takes place using `cmsh localhost`, then the management interface is most probably not up. In that case, bringing the management interface up allows `cmsh` to connect to CMDaemon.
Running `cmsh` without arguments starts an interactive cluster management session. To go back to the unix shell, a user enters `quit` or `ctrl-d`:

```
[mycluster]% quit
[root@mycluster ~]#
```

Batch Mode And Piping In `cmsh`

The `-c` flag allows `cmsh` to be used in batch mode. Commands may be separated using semi-colons:

```
[root@mycluster ~]# cmsh -c "main showprofile; device status apc01"
admin
apc01 ............... [   UP   ]
[root@mycluster ~]#
```

Alternatively, commands can be piped to `cmsh`:

```
[root@mycluster ~]# echo device status | cmsh
device status
apc01 ............... [   UP   ]
mycluster ........... [   UP   ]
node001 ............. [   UP   ]
node002 ............. [   UP   ]
switch01 ............ [   UP   ]
[root@mycluster ~]#
```

Dotfiles And `/etc/cmshrc` File For `cmsh`

In a similar way to unix shells, `cmsh` sources an rc file from the `/etc` directory, and also dotfiles, if they exist. The sourcing is done upon start-up in both batch and interactive mode.

If `/etc/cmshrc` exists, then its settings are used, but the values can be overridden by user dotfiles. This is standard Unix behavior, analogous to how `bash` works with `/etc/bashrc` and `.bashrc` files.

In the following list of `cmsh` dotfiles, a setting in the file that is in the shorter path overrides a setting in the file with the longer path (i.e.: "shortest path overrides"):

- `~/.cm/cmsh/.cmshrc`
- `~/.cm/.cmshrc`
- `~/.cmshrc`

Defining Command Aliases In `cmsh`

Sourcing settings is convenient when defining command aliases. Command aliases can be used to abbreviate longer commands. For example, putting the following in `.cmshrc` would allow `lv` to be used as an alias for `device list virtualnode`:

Example

```
alias lv device list virtualnode
```

Besides defining aliases in dotfiles, aliases in cmsh can also be created with the alias command. The preceding example can be run within cmsh to create the lv alias. Running the alias command within cmsh lists the existing aliases.

Aliases can be exported from within cmsh together with other cmsh dot settings with the help of the export command:

Example

```
[mycluster]% export > /root/mydotsettings
```

The dot settings can be taken into cmsh by running the run command from within cmsh:

Example

```
[mycluster]% run /root/mydotsettings
```

Built-in Aliases In cmsh

The following aliases are built-ins, and are not defined in any .cmshrc or cmshrc files:

```
[basecm11]% alias
alias - goto -
alias .. exit
alias / home
alias ? help
alias ds device status
alias ls list
```

The meanings are:

- goto -: go to previous directory level of cmsh
- exit: go up a directory level, or leave cmsh if already at top level.
- home: go to the top level directory
- help: show help text for current level
- device status: show status of devices that can be accessed in device mode

Automatic Aliases In cmsh

A cmsh script is a file that has a sequence of cmsh commands that run within a cmsh session.

The directory .cm/cmsh/ can have placed in it a cmsh script with a .cmsh suffix and an arbitrary prefix. The prefix then automatically becomes an alias in cmsh.

In the following example

- the file tablelist.cmsh provides the alias tablelist, to list devices using the | symbol as a delimiter, and
- the file dfh.cmsh provides the alias dfh to carry out the Linux shell command df -h

Example

```
[root@mycluster ~]# cat /root/.cm/cmsh/tablelist.cmsh
list -d "|"
[root@mycluster ~]# cat /root/.cm/cmsh/dfh.cmsh
!df -h
[root@mycluster ~]# cmsh
[mycluster]% device
[mycluster->device]% alias | egrep '(tablelist|dfh)'
```

```
alias dfh run /root/.cm/cmsh/dfh.cmsh
alias tablelist run /root/.cm/cmsh/tablelist.cmsh
[mycluster->device]% list
Type                 Hostname (key)   MAC                Category         Ip
-------------------- ---------------- ------------------ ---------------- ---------------
HeadNode             mycluster        FA:16:3E:B4:39:DB                   10.141.255.254
PhysicalNode         node001          FA:16:3E:D5:87:71  default          10.141.0.1
PhysicalNode         node002          FA:16:3E:BE:05:FE  default          10.141.0.2
[mycluster->device]% tablelist
Type                 |Hostname (key)  |MAC               |Category        |Ip
---------------------|----------------|------------------|----------------|---------------
HeadNode             |mycluster       |FA:16:3E:B4:39:DB |                |10.141.255.254
PhysicalNode         |node001         |FA:16:3E:D5:87:71 |default         |10.141.0.1
PhysicalNode         |node002         |FA:16:3E:BE:05:FE |default         |10.141.0.2
[mycluster->device]% dfh
Filesystem      Size  Used Avail Use% Mounted on
devtmpfs        1.8G     0  1.8G   0% /dev
tmpfs           1.9G     0  1.9G   0% /dev/shm
tmpfs           1.9G   33M  1.8G   2% /run
tmpfs           1.9G     0  1.9G   0% /sys/fs/cgroup
/dev/vdb1        25G   17G  8.7G  66% /
tmpfs           374M     0  374M   0% /run/user/0
```

The cmsh session in NVIDIA Base Command Manager does not need restarting for the alias to become active.

Default Arguments In cmsh Scripts

In a cmsh script, the parameters $1, $2 and so on can be used to pass arguments. If the argument being passed is blank, then the values the parameters take also remain blank. However, if the parameter format has a suffix of the form -<*value*>, then <*value*> is the default value that the parameter takes if the argument being passed is blank.

Example

```
[root@mycluster ~]# cat .cm/cmsh/encrypt-node-disk.cmsh
home
device use ${1-node001}
set disksetup /root/my-encrypted-node-disk.xml
set revision ${2-test}
commit
```

The script can be run without an argument (a blank value for the argument), in which case it takes on the default value of node001 for the parameter:

```
[root@mycluster ~]# cmsh
[mycluster]% encrypt-node-disk
[mycluster->device[node001]]%
```

The script can be run with an argument (node002 here), in which case it takes on the passed value of node002 for the parameter:

```
[root@mycluster ~]# cmsh
[mycluster]% encrypt-node-disk node002
[mycluster->device[node002]]%
```

Options Usage For `cmsh`

The options usage information for `cmsh` is obtainable with `cmsh -h`:

```
Usage:
          cmsh [options] [hostname[:port]]
          cmsh [options] -c <command>
          cmsh [options] -f <filename>

Options:

          --help|-h
               Display this help

          --noconnect|-u
               Start unconnected

          --controlflag|-z
               ETX in non-interactive mode

          --color <yes/no>
               Define usage of colors

          --spool <directory>
               Alternative /var/spool/cmd

          --tty|-t
               Pretend a TTY is available

          --noredirect|-r
               Do not follow redirects

          --norc|-n
               Do not load cmshrc file on start-up

          --noquitconfirmation|-Q
               Do not ask for quit confirmation

          --echo|-x
               Echo all commands

          --quit|-q
               Exit immediately after error

          --disablemultiline|-m
               Disable multiline support

          --hide-events
               Hide all events by default

          --disable-events
              Disable all events by default

          --certificate|-i
               Specify alternative certificate

          --key|-k
```

```
               Specify alternative private key

Arguments:
          hostname
              The hostname or IP to connect to

          command
              A list of cmsh commands to execute

          filename
              A file which contains a list of cmsh commands to execute

Examples:
          cmsh                                    run in interactive mode
          cmsh -c 'device status'                 run the device status command and exit
          cmsh --hide-events -c 'device status'   run the device status command and exit, without
                                                  showing any events that arrive during this time
          cmsh -f some.file -q -x                 run and echo the commands from some.file, exit
```

Events can be suppressed in `cmsh`:

- The `--hide-events` flag suppresses the display of events in an interactive `cmsh` session.
- The `--disable-events` flag disables event processing entirely and is intended for non-interactive scripted commands. It is compatible with `--wait` options for commands within a `cmsh` session, allowing a script to suppress event output while still waiting for an operation to complete.

Example

```
[basecm11 ~]# cmsh --disable-events -c "softwareimage use dgx-image; createramdisk --wait"
```

Man Page For `cmsh`

There is also a man page for `cmsh(8)`, which is a bit more extensive than the help text. It does not however cover the modes and interactive behavior.

2.5.2 Levels, Modes, Help, And Commands Syntax In `cmsh`

The *top-level* of `cmsh` is the level that `cmsh` is in when entered without any options.

To avoid overloading a user with commands, cluster management functionality has been grouped and placed in separate `cmsh` *mode* levels. Mode levels and associated *objects* for a level make up a hierarchy available below the top-level.

There is an object-oriented terminology associated with managing via this hierarchy. To perform cluster management functions, the administrator descends via `cmsh` into the appropriate mode and object, and carries out actions relevant to the mode or object.

For example, within `user` mode, an object representing a user instance, `fred`, might be added or removed. Within the object `fred`, the administrator can manage its properties. The properties can be data such as a password `fred123`, or a home directory `/home/fred`.

Figure 2.5 shows the top-level commands available in `cmsh`. These commands are displayed when `help` is typed in at the top-level of `cmsh`:

```
alias ......................... Set aliases
category ...................... Enter category mode
cert .......................... Enter cert mode
cloud ......................... Enter cloud mode
color ......................... Manage console text color settings
configurationoverlay .......... Enter configurationoverlay mode
connect ....................... Connect to cluster
delimiter ..................... Display/set delimiter
device ........................ Enter device mode
disconnect .................... Disconnect from cluster
edgesite....................... Enter edgesite mode
etcd .......................... Enter etcd mode
events ........................ Manage events
exit .......................... Exit from current object or mode
export ........................ Display list of aliases current list formats
fspart .......................  Enter fspart mode
group ......................... Enter group mode
groupingsyntax ................ Manage the default grouping syntax
help .......................... Display this help
hierarchy ..................... Enter hierarchy mode
history ....................... Display command history
kubernetes..................... Enter kubernetes mode
list .......................... List state for all modes
main .......................... Enter main mode
modified ...................... List modified objects
monitoring .................... Enter monitoring mode
network ....................... Enter network mode
nodegroup ..................... Enter nodegroup mode
partition ..................... Enter partition mode
powercircuit .................. Enter powercircuit mode
process ....................... Enter process mode
profile ....................... Enter profile mode
quit .......................... Quit shell
quitconfirmation .............. Manage the status of quit confirmation
rack .......................... Enter rack mode
refresh ....................... Refresh all modes
run ........................... Execute cmsh commands from specified file
session ....................... Enter session mode
softwareimage ................. Enter softwareimage mode
task .......................... Enter task mode
time .......................... Measure time of executing command
unalias ....................... Unset aliases
user .......................... Enter user mode
watch ......................... Execute a command periodically, showing output
wlm ........................... Enter wlm mode
```

Figure 2.5: Top level commands in `cmsh`

All levels inside `cmsh` provide these top-level commands.
Passing a command as an argument to `help` gets details for it:

Example

```
[myheadnode]% help run
Name:     run - Execute all commands in the given file(s)
```

```
Usage:    run [OPTIONS] <filename> [<filename2> ...]

Options:  -x, --echo
               Echo all commands

          -q, --quit
               Exit immediately after error

[myheadnode]%
```

In the general case, invoking `help` at any mode level or within an object, without an argument, provides two lists:

- Firstly, under the title of `Top`: a list of top-level commands.
- Secondly, under the title of the level it was invoked at: a list of commands that may be used at that level.

For example, entering `session` mode and then typing in `help` displays, firstly, output with a title of `Top`, and secondly, output with a title of `session` (some output ellipsized):

Example

```
[myheadnode]% session
[myheadnode->session]% help
============================ Top ============================
alias ......................... Set aliases
category ...................... Enter category mode
cert .......................... Enter cert mode
cloud ......................... Enter cloud mode
...
========================== session ==========================
id ....................... Display current session id
killsession .............. Kill a session
list ..................... Provide overview of active sessions
[myheadnode->session]%
```

Navigation Through Modes And Objects In `cmsh`

The major modes tree is shown in Appendix M.1.

The following notes can help the cluster administrator in navigating the `cmsh` shell:

- To enter a mode, a user enters the mode name at the `cmsh` prompt. The prompt changes to indicate that `cmsh` is in the requested mode, and commands for that mode can then be run.
- To use an object within a mode, the `use` command is used with the object name. In other words, a mode is entered, and an object within that mode is used. When an object is used, the prompt changes to indicate that that object within the mode is now being used, and that commands are applied for that particular object.
- To leave a mode, and go back up a level, the `exit` command is used. Similarly, if an object is in use, the `exit` command exits the object. At the top level, `exit` has the same effect as the `quit` command, that is, the user leaves `cmsh` and returns to the unix shell. The string `..` is an alias for `exit`.
- The `home` command, which is aliased to `/`, takes the user from any mode depth to the top level.

- The `path` command at any mode depth displays a string that can be used as a path to the current mode and object, in a form that is convenient for copying and pasting into `cmsh`. The string can be used in various ways. For example, it can be useful to define an alias in `.cmshrc` (page 48).

  In the following example, the `path` command is used to print out a string. This string makes it easy to construct a `bash` shell command to run a `list` from the correct place within `cmsh`:

  Example

```
[basecm11->configurationoverlay[slurm-client]->roles[slurmclient]]% list
Name (key)
---------------------------
slurmclient
[basecm11->configurationoverlay[slurm-client]->roles[slurmclient]]% path
home;configurationoverlay;use "slurm-client";roles;use slurmclient;
```

  Pasting the string into a `bash` shell, using the `cmsh` command with the `-c` option, and appending the `list` command to the string, replicates the session output of the `list` command:

```
[basecm11 ~]# cmsh -c 'configurationoverlay;use "slurm-client";roles;use slurmclient; list'
Name (key)
---------------------------
slurmclient
```

  The following example shows the path command can also be used inside the `cmsh` session itself for convenience:

  Example

```
[basecm11]% device
[basecm11->device]% list
Type             Hostname (key)  MAC                Category  Ip             Network     Status
---------------- --------------- ------------------ --------- -------------- ----------- ------
EthernetSwitch   switch01        00:00:00:00:00:00            10.141.0.50    internalnet [ UP ]
HeadNode         basecm11        00:0C:29:5D:55:46            10.141.255.254 internalnet [ UP ]
PhysicalNode     node001         00:0C:29:7A:41:78  default   10.141.0.1     internalnet [ UP ]
PhysicalNode     node002         00:0C:29:CC:4F:79  default   10.141.0.2     internalnet [ UP ]
[basecm11->device]% exit
[basecm11]% device
[basecm11->device]% use node001
[basecm11->device[node001]]% path
home;device;use node001;
[basecm11->device[node001]]% home
[basecm11]% home;device;use node001    #copy-pasted from path output earlier
[basecm11->device[node001]]%
```

A command can also be executed in a mode without staying within that mode. This is done by specifying the mode before the command that is to be executed within that node. Most commands also accept arguments after the command. Multiple commands can be executed in one line by separating commands with semi-colons.

A `cmsh` input line has the following syntax:

<mode> <cmd> <arg> ... <arg>; ... ; <mode> <cmd> <arg> ... <arg>

where *<mode>* and *<arg>* are optional. More precisely, using standard command-synopsis notation, the synopsis is:

```
[<mode>] <cmd> [<arg> ...  ]  [; ...  ; [<mode>] <cmd> [<arg> ...  ]]
```

Example

```
[basecm11->network]% device status basecm11; list
basecm11 ............ [   UP   ]
Name (key)    Type      Netmask bits  Base address  Domain name          Ipv6
------------- --------- ------------- ------------- -------------------- ----
externalnet   External  16            192.168.1.0   brightcomputing.com  no
globalnet     Global    0             0.0.0.0       cm.cluster
internalnet   Internal  16            10.141.0.0    eth.cluster
[basecm11->network]%
```

In the preceding example, while in `network` mode, the `status` command is executed in `device` mode on the host name of the head node, making it display the status of the head node. The `list` command on the same line after the semi-colon still runs in `network` mode, as expected, and not in `device` mode, and so displays a list of networks.

Inserting a semi-colon makes a difference, in that the `mode` is actually entered, so that the list displays a list of nodes (some output truncated here for convenience):

Example

```
[basecm11->network]% device; status basecm11; list
basecm11 ................ [   UP   ]
Type          Hostname (key)  MAC                Category   Ip             Network     Status
------------- --------------- ------------------ ---------  -------------- ----------- ------
HeadNode      basecm11        FA:16:3E:C8:06:D1             10.141.255.254 internalnet [ UP ]
PhysicalNode  node001         FA:16:3E:A2:9C:87  default    10.141.0.1     internalnet [ UP ]
[basecm11->device]%
```

2.5.3 Working With Objects

Modes in `cmsh` work with associated groupings of data called *objects*. For instance, `device` mode works with `device` objects, and `network` mode works with `network` objects.

The commands used to deal with objects have similar behavior in all modes. Not all of the commands exist in every mode, and not all of the commands function with an explicit object:

Command	Description
`use`	Use the specified object. I.e.: Make the specified object the *current object*
`add`	Create the object and use it
`assign`	Assign an existing property to an object
`unassign`	Unassign from an object a previously-assigned property
`clear`	Clear the values of the object
`clone`	Clone the object and use it

...continues

...continued

Command	Description
remove	Remove the object
commit	Commit local changes, done to an object, to CMDaemon
refresh	Undo local changes done to the object
list	List all objects at current level
sort	Sort the order of display for the list command
format	Set formatting preferences for list output
foreach	Execute a set of commands on several objects
show	Display all properties of the object
swap	Swap (exchange) the names of two objects
get	Display specified property of the object
set	Set a specified property of the object
clear	Set default value for a specified property of the object.
append	Append a value to a property of the object, for a multi-valued property
removefrom	Remove a value from a specific property of the object, for a multi-valued property
modified	List objects with uncommitted local changes
usedby	List objects that depend on the object
validate	Do a validation check on the properties of the object
exit	Exit from the current object or mode level

Working with objects with these commands is demonstrated with several examples in this section.

Working With Objects: use, exit

Example

```
[mycluster->device]% use node001
[mycluster->device[node001]]% status
node001 ............. [   UP   ]
[mycluster->device[node001]]% exit
[mycluster->device]%
```

In the preceding example, use node001 issued from within device mode makes node001 the *current object*. The prompt changes accordingly. The status command, without an argument, then returns status information just for node001, because making an object the current object makes subsequent commands within that mode level apply only to that object. Finally, the exit command exits the current object level.

Working With Objects: add, commit, remove

The commands introduced in this section have many implicit concepts associated with them. So an illustrative session is first presented as an example. What happens in the session is then explained in order to familiarize the reader with the commands and associated concepts.

Example

```
[mycluster->device]% add physicalnode node100 10.141.0.100
[mycluster->device*[node100*]]% commit
[mycluster->device[node100]]% category add test-category
[mycluster->category*[test-category*]]% commit
[mycluster->category[test-category]]% remove test-category
```

```
[mycluster->category*]% commit
Successfully removed 1 Categories
Successfully committed 0 Categories
[mycluster->category]% device remove node100
[mycluster->category]% device
[mycluster->device*]% commit
Successfully removed 1 Devices
Successfully committed 0 Devices
[mycluster->device]%
```

`add:` The `add` command creates an object within its associated mode, and in `cmsh` the prompt drops into the object level just created. Thus, at the start in the preceding example, within `device` mode, a new object, named `node100`, is added. For this particular object properties can also be set, such as the type (`physicalnode`), and IP address (`10.141.0.100`). The node object level (`[node100*]`) is automatically dropped into from `device` mode when the `add` command is executed. After execution, the state achieved is that the object has been created with some properties. However, it is still in a temporary, modified state, and not yet persistent.

Asterisk tags in the prompt are a useful reminder of a modified state, with each asterisk indicating a tagged object that has an unsaved, modified property. In this case, the unsaved properties are the IP address setting, the node name, and the node type.

The `add` command—syntax notes:

In most modes the `add` command takes only one argument, namely the name of the object that is to be created. However, in `device` mode an extra object-type, in this case `physicalnode`, is also required as argument, and an optional extra IP argument may also be specified. The response to "`help add`" while in `device` mode gives details:

```
[myheadnode->device]% help add
Name:
          add - Create a new device of the given type with specified hostname

Usage:
          add <type> <hostname>
          add cloudnode <hostname> [provider]
          add physicalnode <hostname> [ip] [interface]

Arguments:
          type          chassis, fabricresourcebox, fabricswitch, genericdevice, litenode,
                        cloudnode, dpu, physicalnode, headnode, powerdistributionunit,
                        racksensor, switch, unmanagednode

          interface     eg. ens3, bond0=ens3+ens4
```

`commit:` The `commit` command is a further step that actually saves any changes made after executing a command. In this case, in the second line, it saves the `node100` object with its properties. The asterisk tag disappears for the prompt if settings for that mode level and below have been saved.

Conveniently, the top level modes, such as the `category` mode, can be accessed directly from within this level if the mode is stated before the command. So, stating the mode `category` before running the `add` command allows the specified category `test-category` to be added. Again, the `test-category` object level within `category` mode is automatically dropped into when the `add` command is executed.

The `-w|--wait` option to `commit`:

The commit command by default does not wait for a state change to complete. This means that the prompt becomes available right away. This means that it is not obvious that the change has taken place, which could be awkward if scripting with cmsh for cloning (discussed shortly) a software image (section 2.1.2). The -w|--wait option to the commit command works around this issue by waiting for any associated background task, such as the cloning of a software image, to be completed before making the prompt available.

remove: The remove command removes a specified object within its associated mode. On successful execution, if the prompt is at the object level, then the prompt moves one level up. The removal is not actually carried out fully yet; it is only a proposed removal. This is indicated by the asterisk tag, which remains visible, until the commit command is executed, and the test-category removal is saved. The remove command can also remove a object in a non-local mode, if the non-local mode is associated with the command. This is illustrated in the example where, from within category mode, the device mode is declared before running the remove command for node100. The proposed removal is configured without being made permanent, but in this case no asterisk tag shows up in the category mode, because the change is in device mode. To drop into device mode, the mode command "device" is executed. An asterisk tag then does appear, to remind the administrator that there is still an uncommitted change (the node that is to be removed) for the mode. The commit command would remove the object whichever mode it is in—the non-existence of the asterisk tag does not change the effectiveness of commit.

The -d|--data option to remove:

The remove command by default removes an object, and not the represented data. An example is if, in softwareimage mode, a software image is removed with the remove (without options) command. As far as the cluster manager is concerned, the image is removed after running commit. However the data in the directory for that software image is not removed. The -d|--data option to the remove command arranges removal of the data in the directory for the specified image, as well as removal of its associated object.

The -a|--all option to remove:

The remove command by default does not remove software image revisions. The -a|--all option to the remove command also removes all software image revisions.

Working With Objects: clone, modified, swap

Continuing on with the node object node100 that was created in the previous example, it can be cloned to node101 as follows:

Example

```
[mycluster->device]% clone node100 node101
Warning: The Ethernet switch settings were not cloned, and have to be set manually
[mycluster->device*[node101*]]% exit
[mycluster->device*]% modified
State  Type                      Name
------ ------------------------ ----------------------------------
+      Device                    node101
[mycluster->device*]% commit
[mycluster->device]%
[mycluster->device]% remove node100
[mycluster->device*]% commit
[mycluster->device]%
```

The modified command is used to check what objects have uncommitted changes, and the new object node101 that is seen to be modified, is saved with a commit. The device node100 is then removed by using the remove command. A commit executes the removal.

The "+" entry in the State column in the output of the modified command in the preceding example indicates the object is a newly added one, but not yet committed. Similarly, a "~" entry indicates an object that is to be removed on committing, while a blank entry indicates that the object has been modified without an addition or removal involved.

Cloning an object is a convenient method of duplicating a fully configured object. When duplicating a device object, cmsh will attempt to automatically assign a new IP address using a number of heuristics. In the preceding example, node101 is assigned IP address 10.141.0.101.

The attempt is a best-effort, and does not guarantee a sensibly-configured object. The cluster administrator should therefore inspect the result.

Sometimes an object may have been misnamed, or physically swapped. For example, node001 exchanged physically with node002 in the rack, or the hardware device eth0 is misnamed by the kernel and should be eth1. In that case it can be convenient to simply swap their names via the cluster manager front end rather than change the physical device or adjust kernel configurations. This is equivalent to exchanging all the attributes from one name to the other.

For example, if the two interfaces on the head node need to have their names exchanged, it can be done as follows:

```
[mycluster->device]% use mycluster
[mycluster->device[mycluster]]% interfaces
[mycluster->device[mycluster]->interfaces]% list
Type         Network device name  IP               Network
------------ -------------------- ---------------- --------------
physical     eth0 [dhcp]          10.150.4.46      externalnet
physical     eth1 [prov]          10.141.255.254   internalnet
[basecm11->device[mycluster]->interfaces]% swap eth0 eth1; commit
[basecm11->device[mycluster]->interfaces]% list
Type         Network device name  IP               Network
------------ -------------------- ---------------- --------------
physical     eth0 [prov]          10.141.255.254   internalnet
physical     eth1 [dhcp]          10.150.4.46      externalnet
[mycluster->device[mycluster]->interfaces]% exit; exit
```

Working With Objects: get, set, refresh

The get command is used to retrieve a specified property from an object, and set is used to set it:

Example

```
[mycluster->device]% use node101
[mycluster->device[node101]]% get category
test-category
[mycluster->device[node101]]% set category default
[mycluster->device*[node101*]]% get category
default
[mycluster->device*[node101*]]% modified
State  Type                     Name
------ ----------------------- ------------------------------
       Device                   node101
[mycluster->device*[node101*]]% refresh
[mycluster->device[node101]]% modified
No modified objects of type device
[mycluster->device[node101]]% get category
test-category
[mycluster->device[node101]]%
```

Here, the category property of the node101 object is retrieved by using the get command. The

property is then changed using the set command. Using get confirms that the value of the property has changed, and the modified command reconfirms that node101 has local uncommitted changes.

The refresh command undoes the changes made. The modified command then confirms that no local changes exist. Finally the get command reconfirms that no local change took place.

Among the possible values a property can take on are strings and Booleans:

- A string can be set as a revision label for any object:

  Example

  ```
  [mycluster->device[node101]]% set revision "changed on 10th May"
  [mycluster->device*[node101*]]% get revision
  [mycluster->device*[node101*]]% changed on 10th May 2011
  ```

  This can be useful when using shell scripts with an input text to label and track revisions when sending commands to cmsh. How to send commands from the shell to cmsh is introduced in section 2.5.1.

- For Booleans, the values "yes", "1", "on" and "true" are equivalent to each other, as are their opposites "no", "0", "off" and "false". These values are case-insensitive.

Working With Objects: clear

Example

```
[mycluster->device]% set node101 mac 00:11:22:33:44:55
[mycluster->device*]% get node101 mac
00:11:22:33:44:55
[mycluster->device*]% clear node101 mac
[mycluster->device*]% get node101 mac
00:00:00:00:00:00
[mycluster->device*]%
```

The get and set commands are used to view and set the MAC address of node101 without running the use command to make node101 the *current object*. The clear command then unsets the value of the property. The result of clear depends on the type of the property it acts on. In the case of string properties, the empty string is assigned, whereas for MAC addresses the special value 00:00:00:00:00:00 is assigned.

Working With Objects: list, format, sort

The list command is used to list objects in a mode. The command has many options. The ones that are valid for the current mode can be viewed by running help list. The -f|--format option is available in all modes, and takes a format string as argument. The string specifies what properties are printed for each object, and how many characters are used to display each property in the output line. In following example a list of objects is requested for device mode, displaying the hostname, switchports and ip properties for each device object.

Example

```
[basecm11->device]% list -f hostname:14,switchports:15,ip
hostname (key) switchports     ip
-------------- --------------- --------------------
apc01                          10.142.254.1
basecm11       switch01:46     10.142.255.254
node001        switch01:47     10.142.0.1
node002        switch01:45     10.142.0.2
switch01                       10.142.253.1
[basecm11->device]%
```

Running the `list` command with no argument uses the current format string for the mode. Running the list command with the `-f` option, and without specifying a size for the column keeps the column 20 characters wide.

Running the `format` command without arguments displays the current format string, and also displays all available properties including a description of each property. For example (output truncated):

Example

```
[basecm11->device]% format
Current list printing format:
------------------------------
type:22, hostname:[16-32], mac:18, category:[16-32], ip:15, network:[14-32], status:[16-32]

Valid fields:
-------------
activation                : Date on which node was defined
additionalhostnames       : List of additional hostnames that should resolve to the interfaces IP address
allownetworkingrestart    : Allow node to update ifcfg files and restart networking
banks                     : Number of banks
...
```

The print specification of the `format` command uses the delimiter ":" to separate the parameter and the value for the width of the parameter column. For example, a width of 10 can be set with:

Example

```
[basecm11->device]% format hostname:10
[basecm11->device]% list
hostname (
----------
apc01
basecm11
node001
node002
switch01
```

Setting a width of `0` means that the column takes on the width of the widest entry.

A range of widths can be specified for a column, from a minimum to a maximum, using square brackets. When a range is set, then the column width matches at least the lower bound, and at most the upper bound. The text for the column is fitted into this range. If possible, the column width is shrunk to fit the text that is displayed. If the text is larger than the upper bound, then the text is truncated with a + to that width. For example:

Example

```
[basecm11->device]% format hostname:[10-14]
[basecm11->device]% list
hostname (key)
--------------
apc01
basecm11
node001
node002
switch01
largenameforn+
```

Here a node called largenamefornode is truncated to 14 characters, with the + replacing the 14th character.

The parameters to be viewed can be chosen from a list of valid fields by running the format command without any options, as shown earlier.

The format command can take as an argument a string that is made up of multiple parameters in a comma-separated list. Each parameter takes a colon-delimited width specification.

Example

```
[basecm11->device]% format hostname:[10-14],switchports:14,ip:20
[basecm11->device]% list
hostname (key) switchports    ip
-------------- -------------- --------------------
apc01                         10.142.254.1
basecm11       switch01:46    10.142.255.254
node001        switch01:47    10.142.0.1
node002        switch01:45    10.142.0.2
switch01                      10.142.253.1
```

The output of the format command without arguments shows the current list printing format string, with spaces. This can be used with enclosing quotes (").

In general, the string used in the format command can be set with enclosing quotes ("), or alternatively, with the spaces removed:

Example

```
[basecm11->device]% format "hostname:[16-32], network:[14-32], status:[16-32]"
```

or

```
[basecm11->device]% format hostname:[16-32],network:[14-32],status:[16-32]
```

The default parameter settings can be restored with the -r|--reset option:

Example

```
[basecm11->device]% format -r
[basecm11->device]% format | head -3
Current list printing format:
-----------------------------
type:22, hostname:[16-32], mac:18, category:[16-32], ip:15, network:[14-32], status:[16-32]
[basecm11->device]%
```

The sort command sorts output in alphabetical order for specified parameters when the list command is run. The sort is done according to the precedence of the parameters passed to the sort command:

Example

```
[basecm11->device]% sort type mac
[basecm11->device]% list -f type:15,hostname:15,mac
type            hostname (key)  mac
--------------- --------------- -------------------
HeadNode        basecm11        08:0A:27:BA:B9:43
PhysicalNode    node002         00:00:00:00:00:00
PhysicalNode    log001          52:54:00:DE:E3:6B
[basecm11->device]% sort type hostname
```

```
[basecm11->device]% list -f type:15,hostname:15,mac
type            hostname (key)  mac
--------------- --------------- --------------------
HeadNode        basecm11        08:0A:27:BA:B9:43
PhysicalNode    log001          52:54:00:DE:E3:6B
PhysicalNode    node002         00:00:00:00:00:00

[basecm11->device]% sort mac hostname
[basecm11->device]% list -f type:15,hostname:15,mac
type            hostname (key)  mac
--------------- --------------- --------------------
PhysicalNode    node002         00:00:00:00:00:00
HeadNode        basecm11        08:0A:27:BA:B9:43
PhysicalNode    log001          52:54:00:DE:E3:6B
```

The preceding `sort` commands can alternatively be specified with the `-s|--sort` option to the list command:

```
[basecm11->device]% list -f type:15,hostname:15,mac --sort type,mac
[basecm11->device]% list -f type:15,hostname:15,mac --sort type,hostname
[basecm11->device]% list -f type:15,hostname:15,mac --sort mac,hostname
```

The `--json` option outputs all matching objects in the current mode as a JSON array. Each array element is a full entity JSON object, using the same serialization as `show --json` (page 66) for that object. The list format string is not used when `--json` is specified.

Example

```
[root@basecm11 ~]# cmsh --hide-events -c "device list --json"
[
  {
    "baseType": "Device",
    "childType": "HeadNode",
    "hostname": "basecm11",
    "mac": "FA:16:3E:F9:10:E0",
    ...
  },
  {
    "baseType": "Device",
    "childType": "PhysicalNode",
    "hostname": "node001",
    ...
  }
]

[root@basecm11 ~]# cmsh --hide-events -c "device interfaces basecm11; list --json"
[
  {
    "baseType": "NetworkInterface",
    "childType": "NetworkPhysicalInterface",
    "name": "eth0",
    "ip": "192.168.200.182",
    ...
  },
  {
    "baseType": "NetworkInterface",
    "name": "eth1",
```

```
    "additionalHostnames": [
      "master"
    ],
    ...
  }
]
```

Working With Objects: `append`, `removefrom`

When dealing with a property of an object that can take more than one value at a time—a list of values—the `append` and `removefrom` commands can be used to respectively append to and remove elements from the list. If more than one element is appended, they should be space-separated. The `set` command may also be used to assign a new list at once, overwriting the existing list. In the following example values are appended and removed from the `powerdistributionunits` properties of device `node001`. The `powerdistributionunits` properties represent the list of ports on power distribution units that a particular device is connected to. This information is relevant when power operations are performed on a node. Chapter 4 has more information on power settings and operations.

Example

```
[mycluster->device]% use node001
[mycluster->device[node001]]% get powerdistributionunits
apc01:1
[...device[node001]]% append powerdistributionunits apc01:5
[...device*[node001*]]% get powerdistributionunits
apc01:1 apc01:5
[...device*[node001*]]% append powerdistributionunits apc01:6
[...device*[node001*]]% get powerdistributionunits
apc01:1 apc01:5 apc01:6
[...device*[node001*]]% removefrom powerdistributionunits apc01:5
[...device*[node001*]]% get powerdistributionunits
apc01:1 apc01:6
[...device*[node001*]]% set powerdistributionunits apc01:1 apc01:02
[...device*[node001*]]% get powerdistributionunits
apc01:1 apc01:2
```

Working With Objects: `usedby`

Removing a specific object is only possible if other objects do not have references to it. To help the administrator discover a list of objects that depend on ("use") the specified object, the `usedby` command may be used. In the following example, objects depending on device `apc01` are requested. The `usedby` property of `powerdistributionunits` indicates that device objects `node001` and `node002` contain references to ("use") the object `apc01`. In addition, the `apc01` device is itself displayed as being in the `up` state, indicating a dependency of `apc01` on itself. If the device is to be removed, then the 2 references to it first need to be removed, and the device also first has to be brought to the `CLOSED` state (page 292) by using the `close` command.

Example

```
[mycluster->device]% usedby apc01
Device used by the following:
Type             Name       Parameter
---------------- ---------- ----------------------
Device           apc01      Device is up
Device           node001    powerDistributionUnits
Device           node002    powerDistributionUnits
[mycluster->device]%
```

Working With Objects: `validate`

Whenever committing changes to an object, the cluster management infrastructure checks the object to be committed for consistency. If one or more consistency requirements are not met, then `cmsh` reports the violations that must be resolved before the changes are committed. The `validate` command allows an object to be checked for consistency without committing local changes.

Example

```
[mycluster->device]% use node001
[mycluster->device[node001]]% clear category
[mycluster->device*[node001*]]% commit
Code  Field                    Message
----- ---------------------- --------------------------
1     category                 The category should be set
[mycluster->device*[node001*]]% set category default
[mycluster->device*[node001*]]% validate
All good
[mycluster->device*[node001*]]% commit
[mycluster->device[node001]]%
```

Working With Objects: `show`

The `show` command is used to show the parameters and values of a specific object. For example for the object `node001`, the attributes displayed are (some output ellipsized):

```
[mycluster->device[node001]]% show
Parameter                                Value
---------------------------------------- ---------------------------------------
Hostname                                 node001
IP                                       10.141.0.1
Network                                  internalnet
Revision
Type                                     PhysicalNode
Mac                                      FA:16:3E:3B:D8:23
Use exclusively for                       (category:default)
Category                                 default
Activation                               Thu, 28 Aug 2025 08:41:00 CEST
Rack
Chassis                                  < not set >
...
From template node
Default gateway                          10.141.255.254 (headnode)
Default gateway metric                   0
Switch ports
Interfaces                               <1 in submode>
Static routes                            <0 in submode>
GPU Settings                             <0 in submode>
BMC Settings                             <submode>
...
```

The `--json` option outputs the object as a JSON object rather than a parameter table. The JSON uses the same field names and values as the underlying entity representation. This is the same serialization used for each element of a `list --json` array (page 64).

Example

```
[basecm11->device[basecm11]->interfaces]% show eth1 --json
{
```

```
  "additionalHostnames": [
    "master"
  ],
  "baseType": "NetworkInterface",
  "bringupduringinstall": "NO",
  "childType": "NetworkPhysicalInterface",
  "connectedMode": true,
  "ip": "10.141.255.254",
  "name": "eth1",
  "network": "3c1daada-7000-432c-8198-c2a220ff5ab5",
  "onNetworkPriority": 60,
  "startIf": "ALWAYS",
  "uuid": "62fe5a89-0e4d-46c2-a4e8-2ed50ddbb81b"
}
```


Working With Objects: `assign`, `unassign`

The `assign` and `unassign` commands are analogous to `add` and `remove`. The difference between `assign` and `add` from the system administrator point of view is that `assign` sets an object with settable properties from a choice of existing names, whereas `add` sets an object with settable properties that include the name that is to be given. This makes `assign` suited for cases where multiple versions of a specific object choice cannot be used.

For example,

- If a node is to be configured to be run with particular Slurm settings, then the node can be assigned an `slurmclient` role (section 2.1.5) with the `assign` command. The node cannot be assigned another `slurmclient` role with other Slurm settings at the same time. Only the settings within the assigned Slurm client role can be changed.
- If a node is to be configured to run with added interfaces `eth3` and `eth4`, then the node can have both physical interfaces added to it with the `add` command.

The only place where the `assign` command is currently used within `cmsh` is within the `roles` submode, available under `category` mode, `configurationoverlay` mode, or `device` mode. Within `roles`, `assign` is used for assigning `roles` objects to give properties associated with that role to the category, configuration overlay, or device.

Working With Objects: `import` **For Roles**

The `import` command is an advanced command that works within a role. It is used to clone roles between entities.

A node inherits all roles from the category and configuration overlay it is a part of.

Example

```
[root@basecm11 ~]# cmsh
[basecm11]% device roles node001
[basecm11->device[node001]->roles]% list
Name (key)
------------------------------
[category:default] cgroupsupervisor
[category:default] slurmclient
```

If there is a small change to the default roles to be made, only for `node001`, in `slurmclient`, then the role can be imported from a category or overlay. Importing the role duplicates the object and assigns the duplicated value to `node001`.

This differs from simply assigning a `slurmclient` role to `node001`, because importing provides the values from the category or overlay, whereas assigning provides unset values.

After running `import`, just as for `assign`, changes to the role made at `node001` level stay at that node level, and changes made to the category-level or overlay-level `slurmclient` role are not automatically inherited by the `node001` `slurmclient` role.

Example

```
[basecm11->device[node001]->roles]% import  <TAB><TAB>
backup           failover         monitoring       slurmaccounting
boot             firewall         mqtt             slurmclient
capi             fspart           nginx            slurmserver
...
[basecm11->device[node001]->roles]% import --overlay slurm-client slurmclient

        ...preceding line imports the value from slurm-client overlay to slurmclient role...

[basecm11->device*[node001*]->roles*]% list
Name (key)
------------------------------
[category:default] cgroupsupervisor
slurmclient
[basecm11->device*[node001*]->roles*]% set slurmclient queues node1q
[basecm11->device*[node001*]->roles*]% commit
```

The preceding shows that a list of possible roles is prompted for via tab-completion after having typed `import`, and that the settings from the configuration overlay level are brought into `node001` for the `slurmclient` role. The `slurmclient` values at node level then override any of the overlay level or category level settings, as suggested by the new list output. The Slurm client settings are then the same for `node001` as the settings at the overlay level. The only change made is that a special queue, `node1q`, is configured just for `node001`.

The `import` command in `roles` mode can duplicate any role between any two entities. Options can be used to import from a category (`-c|--category`), or a node (`-n|--node`), or an overlay (`-o|--overlay`), as indicated by its help text (`help import`).

2.5.4 Accessing Cluster Settings

The management infrastructure of BCM is designed to allow cluster partitioning in the future. A cluster partition can be viewed as a virtual cluster inside a real cluster. The cluster partition behaves as a separate cluster while making use of the resources of the real cluster in which it is contained. Although cluster partitioning is not yet possible in the current version of BCM, its design implications do decide how some global cluster properties are accessed through `cmsh`.

In `cmsh` there is a `partition` mode which will, in a future version, allow an administrator to create and configure cluster partitions. Currently, there is only one fixed partition, called `base`. The `base` partition represents the physical cluster as a whole and cannot be removed. A number of properties global to the cluster exist inside the `base` partition. These properties are referenced and explained in remaining parts of this manual.

Example

```
[root@myheadnode ~]# cmsh
[myheadnode]% partition use base
[myheadnode->partition[base]]% show
Parameter                      Value
------------------------------ ------------------------------------------------
Cluster name                   mycluster
Revision
Cluster reference architecture
```

```
Administrator e-mail            gandalf@example.com
Name                            base
Headnode                        myheadnode
Node basename                   node
Node digits                     3
Name servers
Name servers from dhcp          10.3.100.100
Time servers                    0.pool.ntp.org,1.pool.ntp.org,2.pool.ntp.org
Search domains                  example.com
Relay Host
Externally visible IP           0.0.0.0
Time zone                       Europe/Amsterdam
BMC Settings                    <submode>
SNMP Settings                   <submode>
DPU Settings                    <submode>
SELinux Settings                <submode>
Access Settings                 <submode>
Provisioning Settings           <submode>
ZTP settings                    <submode>
ZTP new switch settings         <submode>
NetQ settings                   <submode>
Default burn configuration      default-destructive
External network                externalnet
Management network              internalnet
No zero conf                    no
Default category                default
ArchOS                          <0 in submode>
Fabrics                         <0 in submode>
Sign installer certificates     AUTO
Failover                        not defined
Failover groups                 <0 in submode>
Burn configs                    <3 in submode>
Notes                           <0B>
```

2.5.5 Advanced `cmsh` Features

This section describes some advanced features of `cmsh` and may be skipped on first reading.

Command Line Editing

Command line editing and history features from the `readline` library are available. `http://tiswww.case.edu/php/chet/readline/rluserman.html` provides a full list of key-bindings.

For users who are reasonably familiar with the `bash` shell running with `readline`, probably the most useful and familiar features provided by `readline` within `cmsh` are:

- tab-completion of commands and arguments
- being able to select earlier commands from the command history using `<ctrl>-r`, or using the up- and down-arrow keys

History And Timestamps

The `history` command within `cmsh` explicitly displays the `cmsh` command history as a list.

The `--timestamps|-t` option to the `history` command displays the command history with timestamps.

Example

```
[basecm11->device[node001]]% history | tail -3
```

```
  162 use node001
  163 history
  164 history | tail -3
[basecm11->device[node001]]% history -t | tail -3
  163  Thu Dec  3 15:15:18 2015 history
  164  Thu Dec  3 15:15:43 2015 history | tail -3
  165  Thu Dec  3 15:15:49 2015 history -t | tail -3
```

This history is saved in the file `.cm/.cmshhistory` in the `cmsh` user's directory. The timestamps in the file are in unix epoch time format, and can be converted to human-friendly format with the standard date utility.

Example

```
[root@mycluster ~]# tail -2 .cm/.cmshhistory
1615412046
device list
[root@mycluster ~]# date -d @1615412046
Wed Mar 10 22:34:06 CET 2021
```

Mixing `cmsh` And Unix Shell Commands

It is often useful for an administrator to be able to execute unix shell commands while carrying out cluster management tasks. The cluster manager shell, `cmsh`, therefore allows users to execute commands in a subshell if the command is prefixed with a "`!`" character:

Example

```
[mycluster]% !hostname -f
mycluster.cm.cluster
[mycluster]%
```

Executing the `!` command by itself will start an interactive login sub-shell. By exiting the sub-shell, the user will return to the `cmsh` prompt.

Besides simply executing commands from within `cmsh`, the output of operating system shell commands can also be used within `cmsh`. This is done by using the legacy-style "backtick syntax" available in most unix shells.

Example

```
[mycluster]% device use `hostname`
[mycluster->device[mycluster]]% status
mycluster ................ [   UP   ]
[mycluster->device[mycluster]]%
```

Output Redirection

Similar to unix shells, `cmsh` also supports output redirection to the shell through common operators such as `>`, `>>`, and `|`.

Example

```
[mycluster]% device list > devices
[mycluster]% device status >> devices
[mycluster]% device list | grep node001
Type           Hostname (key)    MAC (key)           Category
-------------- --------------    ------------------- ----------
PhysicalNode   node001           00:E0:81:2E:F7:96   default
```

Input Redirection

Input redirection with cmsh is possible. As is usual, the input can be a string or a file. For example, for a file runthis with some commands stored in it:

Example

```
[root@mycluster ~]# cat runthis
device
get node001 ip
```

the commands can be run with the redirection operator as:

Example

```
[root@mycluster ~]# cmsh < runthis
device
get node001 ip
10.141.0.1
```

Running the file with the -f option avoids echoing the commands

Example

```
[root@mycluster ~]# cmsh -f runthis
10.141.0.1
```

The ssh Command

The ssh command is run from within the device mode of cmsh. If an ssh session is launched from within cmsh, then it clears the screen and is connected to the specified node. Exiting from the ssh session returns the user back to the cmsh launch point.

Example

```
[basecm11]% device ssh node001
<screen is cleared>
<some MOTD text and login information is displayed>
[root@node001 ~]# exit
Connection to node001 closed.
[basecm11]% device use basecm11
[basecm11->device[basecm11]]% #now let us connect to the head node from the head node object
[basecm11->device[basecm11]]% ssh
<screen is cleared>
<some MOTD text and login information is displayed>
[root@basecm11 ~]# exit
logout
Connection to basecm11 closed.
[basecm11->device[basecm11]]%
```

An alternative to running ssh within cmsh is to launch it in a subshell anywhere from within cmsh, by using !ssh.

The time Command

The time command within cmsh is a simplified version of the standard unix time command.

The time command takes as its argument a second command that is to be executed within cmsh. On execution of the time command, the second command is executed. After execution of the time command is complete, the time the second command took to execute is displayed.

Example

```
[basecm11->device]% time ds node001
node001 .................. [   UP   ]
time: 0.108s
```

The `watch` Command

The `watch` command within `cmsh` is a simplified version of the standard unix `watch` command.

The `watch` command takes as its argument a second command that is to be executed within `cmsh`. On execution of the `watch` command, the second command is executed every 2 seconds by default, and the output of that second command is displayed.

The repeat interval of the `watch` command can be set with the `--interval|-n` option. A running `watch` command can be interrupted with a `<Ctrl>-c`.

Example

```
[basecm11->device]% watch newnodes
 screen clears
Every 2.0s: newnodes                                    Thu Dec  3 13:01:45 2015
No new nodes currently available.
```

Example

```
[basecm11->device]% watch -n 3 status -n node001,node002
 screen clears
Every 3.0s: status -n node001,node002                   Thu Jun 30 17:53:21 2016
node001 ..............[ UP ]
node002 ..............[ UP ]
```

Looping Over Objects With `foreach`

It is frequently convenient to be able to execute a `cmsh` command on several objects at once. The `foreach` command is available in a number of `cmsh` modes for this purpose. A `foreach` command takes a list of space-separated object names (the keys of the object) and a list of commands that must be enclosed by parentheses, i.e.: "(" and ")". The `foreach` command will then iterate through the objects, executing the list of commands on the iterated object each iteration.

Basic syntax for the `foreach` command: The basic `foreach` syntax is:

`foreach` *<object1> <object2> ··· (<command1> ; <command2> ···)*

Example

```
[mycluster->device]% foreach node001 node002 (get hostname; status)
node001
node001 ............. [  UP  ]
node002
node002 ............. [  UP  ]
[mycluster->device]%
```

With the `foreach` command it is possible to perform `set` commands on groups of objects simultaneously, or to perform an operation on a group of objects. The `range` command (page 77) provides an alternative to it in many cases.

Advanced options for the `foreach` command: The `foreach` command advanced options can be viewed from the help page:

```
[root@basecm11 ~]# cmsh -c "device help foreach"
```

The options can be classed as: grouping options (list, type), adding options, conditional options, and looping options.

- **Grouping options:**

- -n|--nodes, -g|--group, -c|--category, -r|--rack, -h|--chassis, -e|--overlay, -l|--role, -m|--image, -u|--union, -i|--intersection
- -t|--type chassis|fabricresourcebox|fabricswitch|genericdevice|litenode|cloudnode|dpu|physicalnode|headnode|powerdistributionunit|racksensor|switch|unmanagednode

There are two forms of grouping options shown in the preceding text. The first form uses a list of the objects being grouped, while the second form uses the type of the objects being grouped. These options become available according to the cmsh mode used.

In the device mode of cmsh, for example, the foreach command has many grouping options available. If objects are specifed with a grouping option, then the specified objects can be looped over.

For example, with the list form, the --category (-c) option takes a node category argument (or several categories), while the --node (-n) option takes a node-list argument. Node-lists (specification on page 75) can also use the following, more elaborate, syntax:
<node>,...,*<node>*,*<node>*..*<node>*

Example

```
[demo->device]% foreach -c default (status)
node001 ............. [  DOWN  ]
node002 ............. [  DOWN  ]
[demo->device]% foreach -g rack8 (status)
...
[demo->device]% foreach -n node001,node008..node016,node032 (status)
...
[demo->device]%
```

With the type form, using the -t|--type option, the literal value to this option must be one of node, cloudnode, virtualnode, and so on.

If multiple grouping options are used, then the union operation takes place by default.

Both grouping option forms are often used in commands other than foreach for node selection.

- **Adding options:** -o|--clone, -a|--add
  The --clone (-o) option allows the cloning (section 2.5.3) of objects in a loop. In the following example, from device mode, node001 is used as the base object from which other nodes from node022 up to node024 are cloned:

Example

```
[basecm11->device]% foreach --clone node001 -n node022..node024 ()
[basecm11->device*]% list | grep node
Type          Hostname (key) Ip
------------  -------------- -----------
PhysicalNode  node001        10.141.0.1
PhysicalNode  node022        10.141.0.22
PhysicalNode  node023        10.141.0.23
PhysicalNode  node024        10.141.0.24
[basecm11->device*]% commit
```

To avoid possible confusion: the cloned objects are merely objects (placeholder schematics and settings, with some different values for some of the settings, such as IP addresses, decided by heuristics). So it is explicitly not the software disk image of node001 that is duplicated by object cloning to the other nodes by this action at this time.

– **Overriding the default heuristics for IP address allocation:** The default heuristics for IP address allocation choose the next free IP address if, among other conditions, the same base name is used for the clone. Thus, if the base name used differs from the original, then by default the next free IP address is not chosen. To override the heuristic, so that the next free IP address is chosen anyway, the `--next-ip` option can be used.

For example, when creating nodes starting with `node02` instead of the default `node002`:

Example

```
[basecm11->device]% foreach -o node001 -n node[02-04] ()
Base name mismatch, IP settings will not be modified!
Base name mismatch, IP settings will not be modified!
Base name mismatch, IP settings will not be modified!
[basecm11->device*]% network ips internalnet
Hostname          IP               State
----------------- ---------------- ----------------
basecm11          10.141.255.254   ok
node001           10.141.0.1       duplicate
node02            10.141.0.1       duplicate
node03            10.141.0.1       duplicate
node04            10.141.0.1       ok

[basecm11->device]% foreach -o node001 -n node[02-04] --next-ip ()
[basecm11->device*]% network ips internalnet
Hostname          IP               State
----------------- ---------------- ----------------
basecm11          10.141.255.254   ok
node001           10.141.0.1       ok
node02            10.141.0.2       ok
node03            10.141.0.3       ok
node04            10.141.0.4       ok
```

Conversely, IP addresses can be incremented by a specific amount when using the `addinterface` command (section 3.7.1), by using its `--increment` option.

The `--add` (`-a`) option creates the device for a specified device type, if it does not exist. Valid types are shown in the help output, and include `physicalnode`, `headnode`, `switch`.

- **Conditional options:** `-s|--status`, `-q|--quitonunknown`
  The `--status` (`-s`) option allows nodes to be filtered by the device status (section 2.1.1).

  Example

  ```
  [basecm11->device]% foreach -n node001..node004 --status UP (get IP)
  10.141.0.1
  10.141.0.3
  ```

  Since the `--status` option is also a grouping option, the union operation applies to it by default too, when more than one grouping option is being run.

  The `--quitonunknown` (`-q`) option allows the `foreach` loop to be exited when an unknown command is detected.

- **Looping options:** `*`, `-v|--verbose`
  The wildcard character `*` with `foreach` implies all the objects that the `list` command lists for that mode. It is used without grouping options:

Example

```
[myheadnode->device]% foreach * (get ip; status)
10.141.253.1
switch01 ............ [  DOWN  ]
10.141.255.254
myheadnode .......... [   UP   ]
10.141.0.1
node001 ............. [ CLOSED ]
10.141.0.2
node002 ............. [ CLOSED ]
[myheadnode->device]%
```

Another example that lists all the nodes per category, by running the `listnodes` command within category mode:

Example

```
[basecm11->category]% foreach * (get name; listnodes)
default
Type          Hostname  MAC               Category  Ip         Network      Status
------------- --------- ----------------- --------- ---------- ------------ --------
PhysicalNode  node001   FA:16:3E:79:4B:77  default   10.141.0.1 internalnet  [ UP ]
PhysicalNode  node002   FA:16:3E:41:9E:A8  default   10.141.0.2 internalnet  [ UP ]
PhysicalNode  node003   FA:16:3E:C0:1F:E1  default   10.141.0.3 internalnet  [ UP ]
```

bf The `--verbose` (`-v`) option displays the loop headers during a running loop with time stamps, which can help in debugging.

Restrictions in execution of power-related commands for the `foreach` command: Some power-related commands are prevented from execution in a `foreach` loop.

Example

```
root@basecm11:~# cmsh
[basecm11]% device
[basecm11->device]% foreach -n node001 (power status)
This command is not allowed inside foreach/range: power status
Specify the node selection directly on the command instead
```

This is because the execution time-scale differences between the `foreach` command and the command executed by the `foreach` command can cause issues. The issues can be unacceptable delays in response, or an unacceptable stacking up of powering-on operations. If powering-on operations stack up and cause unfeasibly large power demands, then cloud instances and on-premises instances may not be able to start.

Node List Syntax

Node list specifications, as used in the `foreach` specification and elsewhere, can be of several types. These types are best explained with node list specification examples:

- **adhoc (with a comma, or a space):**
  example: `node001,node003,node005,node006`
- **sequential (with two dots or square brackets):**
  example: `node001..node004`
  or, equivalently: `node00[1-4]`
  which is: `node001,node002,node003,node004`

- **sequential extended expansion (only for square brackets):**
  example: `node[001-002]s[001-005]`
  which is:
  `node001s001,node001s002,node001s003,node001s004,node001s005,\`
  `node002s001,node002s002,node002s003,node002s004,node002s005`

- **rack-based:**
  This is intended to hint which rack a node is located in. Thus:

  – example: `r[1-2]n[01-03]`
    which is: `r1n01,r1n02,r1n03,r2n01,r2n02,r2n03`
    This might hint at two racks, r1 and r2, with 3 nodes each.

  – example: `rack[1-2]node0[1-3]`
    which is: `rack1node01,rack1node02,rack1node03,rack2node01,`
    `rack2node02,rack2node03`
    Essentially the same as the previous one, but for nodes that were named more verbosely.

- **sequential exclusion (negation):**
  example: `node001..node005,-node002..node003`
  which is: `node001,node004,node005`

- **sequential stride (every <*stride*> steps):**
  example: `node00[1..7:2]`
  which is: `node001,node003,node005,node007`

- **mixed list:**
  The square brackets and the two dots input specification cannot be used at the same time in one argument. Other than this, specifications can be mixed:

  – example: `r1n001..r1n003,r2n003`
    which is: `r1n001,r1n002,r1n003,r2n003`

  – example: `r2n003,r[3-5]n0[01-03]`
    which is: `r2n003,r3n001,r3n002,r3n003,r4n001,r4n002,r4n003,r5n001,r5n002,r5n003`

  – example: `node[001-100],-node[004-100:4]`
    which is: every node in the 100 nodes, except for every fourth node.

- **path to file that contains a list of nodes:**
  example: `^/some/filepath/`<*file with list of nodes*>
  The caret sign is a special character in `cmsh` for node list specifications. It indicates the string that follows is a file path that is to be read.

Node list syntax is a customized subset of device list syntax. As such, when using devices other than nodes, some of the syntax from node list syntax may not work as expected.

Setting grouping syntax with the `groupingsyntax` command: "Grouping syntax" here refers to usage of dots and square brackets. In other words, it is syntax of how a grouping is marked so that it is accepted as a list. The list that is specified in this manner can be for input or output purposes.

The `groupingsyntax` command sets the grouping syntax using the following options:

- `bracket`: the square brackets specification.
- `dot`: the two dots specification.
- `auto`: the default. Setting `auto` means that:

- either the dot or the bracket specification are accepted as input,
- the dot specification is used for output.

The chosen groupingsyntax option can be made persistent by adding it to the .cmshrc dotfiles, or to /etc/cmshrc (section 2.5.1).

Example

```
[root@basecm11 ~]# cat .cm/cmsh/.cmshrc
groupingsyntax auto
```

The range Command

The range command provides an interactive option to carry out basic foreach commands over a grouping of nodes. When the grouping option has been chosen, the cmsh prompt indicates the chosen range within braces ({}).

Example

```
[basecm11->device]% range -n node0[01-24]
[basecm11->device{-n node001..024}]%
```

In the preceding example, commands applied at device level will be applied to the range of 24 node objects.

Continuing the preceding session—if a category can be selected with the -c option. If the default category just has three nodes, then output displayed could look like:

Example

```
[basecm11->device{-n node001..024}]% range -c default
[basecm11->device{-c default}]% ds
node001 .................. [   UP   ] state flapping
node002 .................. [   UP   ]
node003 .................. [   UP   ]
```

Values can be set at device mode level for the selected grouping.

Example

```
[basecm11->device{-c default}]% get revision




[basecm11->device{-c default}]% set revision test
[basecm11->device{-c default}]% get revision
test
test
test
```

Values can also be set within a submode. However, staying in the submode for a full interaction is not possible. The settings must be done by entering the submode using a semi-colon (new command statement continuation on same line) syntax, as follows:

Example

```
[basecm11->device{-c default}]% roles; assign slurmclient; commit
```

The range command can be regarded as a modal way to carry out an implicit foreach on the grouping object. Many administrators should find it easier than a foreach:

Example

```
[basecm11->device{-c default}]% get ip
10.141.0.1
10.141.0.2
10.141.0.3
[basecm11->device{-c default}]% ..
[basecm11->device]% foreach -c default (get ip)
10.141.0.1
10.141.0.2
10.141.0.3
```

Commands can be run inside a range. However, running a pexec command inside a range is typically not the intention of the cluster administrator, even though it can be done:

Example

```
[basecm11->device]% range -n node[001-100]
[basecm11->device{-n node[001-100]}]% pexec -n node[001-100] hostname
```

The preceding starts 100 pexec commands, each running on each of the 100 nodes.

Further options to the range command can be seen with the help text for the command (output truncated):

Example

```
[root@basecm11 ~]# cmsh -c "device help range"
Name:     range - Set a range of several devices to execute future commands on

Usage:    range [OPTIONS] * (command)
          range [OPTIONS] <device> [<device> ...] (command)

Options:      --show     Show the current range
              --clear    Clear the range
          -v, --verbose  Show header before each element
...
```

The bookmark And goto Commands

Bookmarks: A *bookmark* in cmsh is a location in the cmsh hierarchy.

A bookmark can be

- set with the bookmark command
- reached using the goto command

A bookmark is set with arguments to the bookmark command within cmsh as follows:

- The user can set the current location as a bookmark:
  - by using no argument. This is the same as setting no name for it
  - by using an arbitrary argument. This is the same as setting an arbitrary name for it
- Apart from any user-defined bookmark names, cmsh automatically sets the special name: "-". This is always the previous location in the cmsh hierarchy that the user has just come from.

All bookmarks that have been set can be listed with the -l|--list option.

Reaching a bookmark: A bookmark can be reached with the `goto` command. The `goto` command can take the following as arguments: a blank (no argument), any arbitrary bookmark name, or "-". The bookmark corresponding to the chosen argument is then reached.

The "-" bookmark does not need to be preceded by a `goto`.

Example

```
[mycluster]% device use node001
[mycluster->device[node001]]% bookmark
[mycluster->device[node001]]% bookmark -l
Name             Bookmark
---------------- -----------------------
                 home;device;use node001;
-                home;
[mycluster->device[node001]]% home
[mycluster]% goto
[mycluster->device[node001]]% goto -
[mycluster]% goto
[mycluster->device[node001]]% bookmark dn1
[mycluster->device[node001]]% goto -
[mycluster]% goto dn1
[mycluster->device[node001]]%
```

Saving bookmarks, and making them persistent: Bookmarks can be saved to a file, such as `mysaved`, with the `-s|--save` option, as follows:

Example

```
[mycluster]% bookmark -s mysaved
```

Bookmarks can be made persistent by setting `(.)cmshrc` files (page 48) to load a previously-saved bookmarks file whenever a new `cmsh` session is started. The `bookmark` command loads a saved bookmark file using the `-x|--load` option.

Example

```
[root@basecm11 ~]# cat .cm/cmsh/.cmshrc
bookmark -x mysaved
```

Renaming Nodes With The `rename` Command

Nodes can be renamed globally from within `partition` mode, in the `Node basename` field associated with the prefix of the node in Base View (section 3.1.1) or in `cmsh` (section 2.5.4, and also page 115).

However, a more fine-grained batch renaming is also possible with the `rename` command, and typically avoids having to resort to scripting mechanisms. Using `rename` is best illustrated by examples:

The examples begin with using the default basename of `node` and default node digits (padded suffix number length) of 3.

A simple rename that is a prefix change, can then be carried out as:

Example

```
[basecm11->device]% rename node001..node003 test
Renamed: node001 to test1
Renamed: node002 to test2
Renamed: node003 to test3
```

The rename starts up its own numbering from 1, independent of the original numbering. The change is committed using the `commit` command.

Zero-padding occurs if the number of nodes is sufficiently large to need it. For example, if 10 nodes are renamed (some output elided):

Example

```
[basecm11->device]% rename node[001-010] test
Renamed: node001 to test01
Renamed: node002 to test02
...
Renamed: node009 to test09
Renamed: node010 to test10
```

then 2 digits are used for each number suffix, in order to match the size of the last number.

String formatting can be used to specify the number of digits in the padded number field:

Example

```
[basecm11->device]% rename node[001-003] test%04d
Renamed: node001 to test0001
Renamed: node002 to test0002
Renamed: node003 to test0003
```

The target names can conveniently be specified exactly. It requires an exact name mapping. That is, it assumes the source list size and target list size match:

Example

```
[basecm11->device]% rename node[001-005] test0[1,2,5-7]
Renamed: node001 to test01
Renamed: node002 to test02
Renamed: node003 to test05
Renamed: node004 to test06
Renamed: node005 to test07
```

The hostnames are sorted alphabetically before they are applied, with some exceptions based on the listing method used.

A `--dry-run` option can be used to show how the devices will be renamed. Alternatively, the `refresh` command can clear a proposed set of changes before a `commit` command commits the change, although the refresh would also remove other pending changes.

Exact name mapping could be used to allocate individual servers to several people:

Example

```
[root@basecm11 ~]# cmsh
[basecm11]% device
[basecm11->device]% rename node[001-004] alice,bob,charlie,dave
Renamed: node001 to alice
Renamed: node002 to bob
Renamed: node003 to charlie
Renamed: node004 to dave
[basecm11->device]% commit
```

Skipping by a number of nodes is possible using a colon (`:`). An example might be to skip by two so that twin servers can be segregated into left/right.

Example

```
[root@basecm11 ~]# cmsh
[basecm11]% device
[basecm11->device]% rename node[001-100:2] left[001-050]
Renamed: node001 to left001
Renamed: node003 to left002
...
Renamed: node097 to left049
Renamed: node099 to left050
[basecm11->device]% rename node[002-100:2] right[001-050]
Renamed: node002 to right001
Renamed: node004 to right002
...
Renamed: node098 to right049
Renamed: node100 to right050
[basecm11->device]% commit
```

Using CMDaemon Environment Variables In Scripts

Within `device` mode, the `environment` command shows the CMDaemon environment variables (section 3.3.2 of the *Developer Manual*) that can be passed to scripts for particular device.

Example

```
[mycluster->device]% environment node001
Key                                          Value
-------------------------------------------- ------------------------------------------------------------------
CMD_ACTIVE_MASTER_IP                         10.141.255.254
CMD_CATEGORY                                 default
CMD_CLUSTERNAME                              mycluster
CMD_DEVICE_TYPE                              ComputeNode
CMD_ENVIRONMENT_CACHE_EPOCH_MILLISECONDS     1615465821582
CMD_ENVIRONMENT_CACHE_UPDATES                4
...
```

The environment variables can be prepared for use in Bash scripts with the `-e|--export` option:

Example

```
[mycluster->device]% environment -e node001
export CMD_ENVIRONMENT_CACHE_UPDATES=4
export CMD_CATEGORY=default
export CMD_SOFTWAREIMAGE=default-image
export CMD_DEVICE_TYPE=ComputeNode
export CMD_ROLES=
export CMD_FSMOUNT__SLASH_home_FILESYSTEM=nfs
export CMD_NODEGROUPS=
...
```

Creating JSON Format Output From A Table Format Output In `cmsh`

The `list --json` and `show --json` options (pages 64 and 66) output complete entity objects as JSON. The approach described in this section is different: it converts the columns selected by the `-f` format string into JSON objects whose keys are the column headings.

A list of table entries can be converted to a JSON representation by using the delimiter specification option: `-d {}`

By default, the indentation value used is 2. Other values can be set by putting the value inside the braces.

Example

```
[basecm11->device]% list -f hostname,ip,mac,status
hostname (key)       ip                   mac                  status
-------------------- -------------------- -------------------- --------------------
node001              10.141.0.1           FA:16:3E:95:80:9F    [   UP   ]
basecm11             10.141.255.254       FA:16:3E:D3:56:E0    [   UP   ]


[basecm11->device]% color off; list -f hostname,ip,mac,status -d {}
[
  {
    "hostname (key)": "basecm11",
    "ip": "10.141.255.254",
    "mac": "FA:16:3E:D3:56:E0",
    "status": "[   UP   ]"
  },
  {
    "hostname (key)": "node001",
    "ip": "10.141.0.1",
    "mac": "FA:16:3E:95:80:9F",
    "status": "[   UP   ]"
  }
]
[basecm11->device]%
```

The `color off` setting is needed to remove the default console coloring. If the command is to run from the `bash` shell, the same output can be achieved with:

Example

```
[root@basecm11 ~]# cmsh --color=no -c "device; list -f hostname,ip,mac,status -d {}"
```

2.6 Cluster Management Daemon

The *cluster management daemon* or *CMDaemon* is a server process that runs on all nodes of the cluster (including the head node). The cluster management daemons work together to make the cluster manageable. When applications such as `cmsh` and Base View communicate with the cluster, they are actually interacting with the cluster management daemon running on the head node. Cluster management applications never communicate directly with cluster management daemons running on non-head nodes.

The CMDaemon application starts running on any node automatically when it boots, and the application continues running until the node shuts down. Should CMDaemon be stopped manually for whatever reason, its cluster management functionality becomes unavailable, making it hard for administrators to manage the cluster. However, even with the daemon stopped, the cluster remains fully usable for running computational jobs using a workload manager.

The only route of communication with the cluster management daemon is through TCP port `8081`. The cluster management daemon accepts only SSL connections, thereby ensuring all communications are encrypted. Authentication is also handled in the SSL layer using client-side `X509v3` certificates (section 2.3).

On the head node, the cluster management daemon uses a MySQL database server to store all of its internal data. Raw monitoring data, on the other hand, is stored as binary data outside of the MySQL database (section 14.8).

2.6.1 Managing And Inspecting The Cluster Management Daemon

Using `systemctl` To Manage The Cluster Management Daemon

It may be useful to shut down or restart the cluster management daemon. For instance, a restart may be necessary to activate changes when the cluster management daemon configuration file is modified.

The cluster management daemon operation can be controlled through the `systemctl` unit command options:

Example

```
[root@basecm11 etc]# systemctl stop cmd.service
[root@basecm11 etc]# systemctl is-enabled cmd.service
enabled
[root@basecm11 etc]# systemctl status cmd.service
* cmd.service - BCM daemon
     Loaded: loaded (/usr/lib/systemd/system/cmd.service; enabled; preset: enabled)
     Active: inactive (dead) since Tue 2025-04-08 15:44:00 CEST; 48s ago
...
[root@basecm11 etc]# systemctl start cmd.service
[root@basecm11 etc]# systemctl status cmd.service
* cmd.service - BCM daemon
     Loaded: loaded (/usr/lib/systemd/system/cmd.service; enabled; preset: enabled)
     Active: active (running) since Tue 2025-04-08 09:18:18 CEST; 6h ago
...
```

The cluster management daemon can be restarted on all regular nodes that are up:

Example

```
[root@mycluster ~]# pdsh -a "systemctl restart cmd; systemctl is-active cmd"
node001: active
node002: active
node003: active
[root@mycluster ~]#
```

This uses `pdsh`, the parallel shell command (section 14.1).

Using `cmdaemonctl` To Inspect The Cluster Management Daemon

The `cmdaemonctl` command is a way to run some CMDaemon-specific service-related commands:

Example

```
[root@basecm11 etc]# cmdaemonctl -h
cmdaemonctl [OPTIONS...] COMMAND ...

Query or send control commands to the cluster manager daemon.

  -h --help              Show this help

Commands:
  debugon                Turn on CMDaemon debug
  debugoff               Turn off CMDaemon debug
  full-status            Display CMDaemon status
  upgrade                Upgrade CMDaemon database
  logconf                Reload log configuration
[root@basecm11 etc]# cmdaemonctl full-status

CMDaemon version 3.0 is running (active)
Running locally
Debug logging is disabled
```

```
Current Time: Tue, 08 Apr 2025 16:15:41 CEST
Startup Time: Tue, 08 Apr 2025 15:55:39 CEST
Uptime: 20m 1s

CPU Usage: 5.55887u 1.93049s (0.6%)
Memory Usage: 176MB

Sessions Since Startup: 5
Active Sessions: 5

Number of occupied worker-threads: 1
Number of free worker-threads: 14

Connections handled: 323
Requests processed: 323
Total read: 93KB
Total written: 317KB

Average request rate: 16.1 requests/m
Average bandwidth usage: 264.35B/s
```

For performance reasons, CMDaemon should not normally be run in debug mode.

2.6.2 Configuring The Cluster Management Daemon

Many cluster configuration changes can be done by modifying the cluster management daemon configuration file. For the head node, the file is located at:

```
/cm/local/apps/cmd/etc/cmd.conf
```

For regular nodes, it is located inside of the software image that the node uses.

Appendix C describes the supported configuration file directives and how they can be used. Normally there is no need to modify the default settings.

After modifying the configuration file, the cluster management daemon must be restarted to activate the changes.

2.6.3 CMDaemon Versions

Updating CMDaemon

CMDaemon can be updated on the head node with a package manager command such as:

```
yum update cmdaemon
```

and on a regular node image with a command such as:

```
yum update --installroot=/cm/images/<software image> cmdaemon
```

Updating software on the cluster is covered in greater detail in Chapter9.

CMDaemon Version Extraction

For debugging an issue, knowing the version of CMDaemon that is in use on the cluster can be helpful. The `cmdaemonversions` command runs within the `device` mode of `cmsh`. It lists the CMDaemon version running on the nodes of the cluster

Example

```
[basecm11->device]% cmdaemonversions
Hostname         Version index Version hash
```

```
---------------- ------------ ------------
basecm11         146,965      e6f593b676
node001          146,965      e6f593b676
node002          146,965      e6f593b676
```

A higher version index value indicates a more recent CMDaemon version.

The `--join` option is a formatting option that gathers together versions with the same value:

```
[basecm11->device]% cmdaemonversions --join
Version index Version hash Count        Hostnames
------------- ------------ ------------ -------------------------
146,962       dcb306b592   2            node004..node006
146,963       db3a48968c   1            node003
146,965       e6f593b676   3            basecm11,node001..node002
```

A cluster administrator is expected to keep the same version for all nodes, but sometimes updating nodes to the same version can take some time—for example, when waiting for a long job to finish on a node, before rebooting that node.

2.6.4 Configuring The Cluster Management Daemon Logging Facilities

CMDaemon generates log messages in `/var/log/cmdaemon` from specific internal subsystems, such as Workload Management, Service Management, Monitoring, Certs, and so on. By default, none of those subsystems generate detailed (debug-level) messages, as that would make the log file grow rapidly.

CMDaemon Logging Configuration Global Debug Mode

A global debug mode can be enabled in CMDaemon using `cmdaemonctl`:

Example

```
[root@basecm11 ~]# cmdaemonctl -h
cmdaemonctl [OPTIONS...] COMMAND ...

Query or send control commands to the cluster manager daemon.

  -h --help              Show this help

Commands:
  debugon                Turn on CMDaemon debug
  debugoff               Turn off CMDaemon debug
...

[root@basecm11 ~]# cmdaemonctl debugon
CMDaemon debug level on
```

Stopping debug level logs from running for too long by executing `cmdaemonctl debugoff` is a good idea, especially for production clusters. This is important in order to prevent swamping the cluster with unfeasibly large logs.

CMDaemon Subsystem Logging Configuration Debug Mode

CMDaemon subsystems can generate debug logs separately per subsystem, including by severity level. This can be done by modifying the logging configuration file at:

`/cm/local/apps/cmd/etc/logging.cmd.conf`

Within this file, a section with a title of `#Available Subsystems` lists the available subsystems that can be monitored. These subsystems include `MON` (for monitoring), `DB` (for database), `HA` (for high availability), `CERTS` (for certificates), and so on.

CMDaemon Subsystem Logging Configuration Severity Levels

The `debug` setting is one of several severity levels. Other severity levels are `info`, `warning`, `error`, and `all`.

Further details on setting subsystem options are given within the `logging.cmd.conf` file.

For example, to set CMDaemon log output for Monitoring, at a severity level of `warning`, the file contents for the section `severity` might look like:

Example

```
Severity {
   warning: MON
}
```

CMDaemon Subsystem Logging Configuration Deployment

The new logging configuration can be reloaded from the file by restarting CMDaemon:

Example

```
[root@basecm11 etc]# systemctl restart cmd
```

or by triggering it using the event bucket (page 665)

Example

```
[root@basecm11 etc]# echo LOGGING.RELOAD.CONFIG > /var/spool/cmd/eventbucket
```

2.6.5 Configuration File Modification, And The `FrozenFile` Directive

As part of its tasks, the cluster management daemon modifies a number of system configuration files. Some configuration files are completely replaced, while other configuration files only have some sections modified. Appendix A lists all system configuration files that are modified.

- A file that has been generated entirely by the cluster management daemon contains a header:

  ```
  # This file was automatically generated by cmd. Do not edit manually!
  ```

  Such a file will be entirely overwritten, unless the `FrozenFile` configuration file directive (Appendix C, page 920) is used to keep it frozen.

- A file that has had only a section of it generated by the cluster management daemon contains a header and ending sections in the following format:

  ```
  # This section of this file was automatically generated by cmd. Do not edit manually!
  # BEGIN AUTOGENERATED SECTION -- DO NOT REMOVE
  ...
  # END AUTOGENERATED SECTION   -- DO NOT REMOVE
  ```

  Such a file has only the auto-generated sections entirely overwritten, unless the `FrozenFile` configuration file directive is used to keep these sections frozen.

The `FrozenFile` configuration file directive in `cmd.conf` is set as suggested by this example:

Example

```
FrozenFile = { "/etc/dhcpd.conf", "/etc/postfix/main.cf" }
```

If the generated full file or generated section of a file has a manually modified part, and `FrozenFile` is not in use, then during overwriting an event is generated, and the original manually modified configuration file is backed up to:

```
/var/spool/cmd/saved-config-files
```

Using `FrozenFile` can be regarded as a configuration technique (section 3.20.3), and one of various possible configuration techniques (section 3.20.1).

2.6.6 Configuration File Conflicts Between The Standard Distribution And BCM For Generated And Non-Generated Files

While BCM changes as little as possible of the standard distributions that it manages, there can sometimes be unavoidable issues. In particular, sometimes a standard distribution utility or service generates a configuration file that conflicts with what the configuration file generated by BCM carries out (Appendix A).

For example, the Red Hat security configuration tool `system-config-securitylevel` can conflict with what `shorewall` (section 7.2 of the *Installation Manual*) does, while the Red Hat Authentication Configuration Tool `authconfig` (used to modify the `/etc/pam.d/system-auth` file) can conflict with the configuration settings set by BCM for LDAP and PAM.

In such a case the configuration file generated by BCM must be given precedence, and the generation of a configuration file from the standard distribution should be avoided. Sometimes using a fully or partially frozen configuration file (section 2.6.5) allows a workaround. Otherwise, the functionality of the BCM version usually allows the required configuration function to be implemented.

Details on the configuration files installed and updated by the package management system, for files that are "non-generated" (that is, not of the kind in section 2.6.5 or in the lists in Appendixes A.1 and A.2.3), are given in Appendix A.3.

2.6.7 CMDaemon Lite

Introduction

As an alternative to the regular CMDaemon, BCM provides a lightweight CMDaemon, called CMDaemon Lite. This is intended as a minimal alternative to the regular CMDaemon for nodes that are not managed by CMDaemon. CMDaemon Lite is contained in the package `cm-lite-daemon`.

It can be installed on a device where the administrator considers the option of installing a regular, full-featured, CMDaemon to be overkill, but still wants an alternative that allows some basic monitoring, and (if available) GNSS measurements, to be carried out on the device.

CMDaemon Lite is a Python service, and can be run on a device such as a standalone desktop, running Windows, Linux, or MacOS. It uses up one node license per node that it is run on. It requires Python 3.7 or later.

CMDaemon Lite with the standard number of metrics is about 25% lighter on memory resources, and 50% lighter on CPU resources, than the regular CMDaemon.

Deployment

Deployment is typically carried out using the head node to place the zipped CMDaemon Lite packages onto the *lite node*. The lite node is the device that is to run `cm-lite-daemon`.

The zipped packages for the head node: can be picked up with:

Example

```
[root@basecm11 ~]# yum install cm-lite-daemon-dist    #apt install cm-lite-daemon-dist   for Ubuntu
```

The package `cm-lite-daemon` is also available from the APT repositories, instead of `cm-lite-daemon-dist`. It should be used for Cumulus OS and for NVOS.

The package manager places the zip file and bootstrap certificate files under `/cm/shared/apps/cm-lite-daemon-dist/`:

Example

```
[root@basecm11 ~]# ls -al /cm/shared/apps/cm-lite-daemon-dist/
total 212
drwxr-xr-x  2 root root     74 Mar 30 02:38 .
drwxr-xr-x 32 root root   4096 Mar 18 09:57 ..
```

```
-rw-------  1 root root   1704 Mar 30 02:38 bootstrap.key
-rw-------  1 root root   1289 Mar 30 02:38 bootstrap.pem
-rw-r--r--  1 root root 202448 Feb 24 15:19 cm-lite-daemon.zip
```

Placing the files on the lite node: In the following session the lite node is assumed to be running Ubuntu. The session details may differ a bit for other operating systems.

Example

```
[root@basecm11 ~]# scp /cm/shared/apps/cm-lite-daemon-dist/* root@lite01:/opt/
[root@basecm11 ~]# ssh root@lite01
...
root@lite01:~# cd /opt
root@lite01:/opt# unzip cm-lite-daemon.zip
...
root@lite01:/opt# cd cm-lite-daemon
root@lite01:/opt/cm-lite-daemon# ls -l --group-directories-first
total 60
drwxr-xr-x 7 root root 4096 Oct 31  2017 cm_lite_daemon
drwxr-xr-x 2 root root   87 Oct 31  2017 etc
drwxr-xr-x 2 root root   76 Oct 31  2017 examples
drwxr-xr-x 2 root root    6 Oct 31  2017 log
drwxr-xr-x 2 root root  110 Oct 31  2017 service
-rwxr-xr-x 1 root root 7752 Oct 31  2017 cm-lite-daemon
-rwxr-xr-x 1 root root  916 Oct 31  2017 cm-lite-daemon_ctl
-rwxr-xr-x 1 root root 3832 Oct 31  2017 connection_test
-rwxr-xr-x 1 root root  382 Oct 31  2017 install-required-pip-packages
-rwxr-xr-x 1 root root  182 Oct 31  2017 install-required-pip-packages.bat
-rwxr-xr-x 1 root root 8409 Oct 31  2017 register_node
-rwxr-xr-x 1 root root 4683 Oct 31  2017 request_certificate
-rw-r--r-- 1 root root  107 Oct 31  2017 requirements.txt
-rwxr-xr-x 1 root root 5409 Oct 31  2017 unregister_node
```

The `bootstrap.*` files that were also copied over to the lite node should also be moved into the `/opt/cm-lite-daemon/etc/` directory:

Example

```
root@lite01:/opt/cm-lite-daemon# mv ../bootstrap.* /opt/cm-lite-daemon/etc/
```

A virtual Python environment on the lite node: is set up to provide consistency for the configuration scripts. For Ubuntu:

`apt install python3-venv`

should install the `venv` module if it is not already available.

The virtual environment directory for CMDaemon Lite, `venv` (the last argument in the following `python3 -m` line), can then be configured and activated by running:

Example

```
root@lite01:/opt/cm-lite-daemon# python3 -m venv venv
root@lite01:/opt/cm-lite-daemon# source venv/bin/activate
```

The installation requirements for the `cm-lite-daemon` software are then met with:

Example

```
(venv) root@lite01:/opt/cm-lite-daemon# pip install -r requirements.txt
Collecting pyOpenSSL>=21.0.0 (from -r requirements.txt (line 1))
  Downloading pyopenssl-26.0.0-py3-none-any.whl.metadata (18 kB)
Collecting websocket-client (from -r requirements.txt (line 2))
  Downloading websocket_client-1.9.0-py3-none-any.whl.metadata (8.3 kB)
...
(venv) [root@lite01 cm-lite-daemon]#
```

Registering the lite node with CMDaemon on the head node: can be carried out on the lite node with:

Example

```
(venv) root@lite01:/opt/cm-lite-daemon# ./register_node --node lite01 --disable-cert-check \
 --disable-hostname-check
INFO     ...    Certificate requested: ec53aa08-d796-4f95-9e24-088e2f81cb90, waiting for it to be issued...
INFO     ...    Saved private key: /opt/cm-lite-daemon/etc/cert.key
INFO     ...    Saved certificate: /opt/cm-lite-daemon/etc/cert.pem
...
(venv) root@lite01:/opt/cm-lite-daemon#
```

If the lite node is on the same network: as the regular nodes being managed by regular CMDaemon on the head node, then the CMDaemon Lite certificate is automatically issued for the lite node by the regular CMDaemon on the head node.

In this case, CMDaemon Lite is configured to run automatically on startup, and its status should show up the lite node as:

Example

```
(venv) root@lite01:/opt/cm-lite-daemon# systemctl status cm-lite-daemon.service
  cm-lite-daemon.service - Bright Computing Cluster Lite Manager daemon
     Loaded: loaded (/usr/lib/systemd/system/cm-lite-daemon.service; enabled; preset: enabled)
     Active: active (running) since Mon 2026-03-30 11:22:29 PDT; 1min 7s ago
   Main PID: 11872 (python3)
      Tasks: 22 (limit: 128)
     Memory: 28.7M (high: 488.0M max: 512.0M available: 459.2M peak: 54.0M)
        CPU: 1.656s
     CGroup: /system.slice/cm-lite-daemon.service
             └─11872 python3 /opt/cm-lite-daemon/cm-lite-daemon --daemon --pid-file /var/run/cm-lite-daemon.pid

Mar 30 11:22:29 lite01 systemd[1]: Started cm-lite-daemon.service - Bright Computing Cluster Lite Manager daemon.
```

If the lite node is on another network: compared to the regular nodes being managed by regular CMDaemon on the head node, then the CMDaemon Lite certificate must be issued manually for the lite node by the regular CMDaemon on the head node. This can be done as follows:

Using `cmsh` the certificate request ID can be found:

Example

```
[basecm11->cert]% listrequests
Request ID   Client type  Session ID   Name
------------ ------------ ------------ ------------
1            Lite node    ....         ....
```

After finding the correct value for the `Request ID`, the certificate can then be issued. For a certificate with a `Request ID` value of `1`, it can be issued with, for example:

```
basecm11->cert]% issuecertificate --days 10000 1
```

The days field can be used to set how long `cm-lite-daemon` is allowed to connect. Regular BCM node certificates have a lifetime of about 10,000 days (about 27 years).

The network of the lite node must also be set to be explicitly managed by CMDaemon. For example: if a lite node, `lite02`, is on another network `internalnet2` that is not the management network, then it needs to have its management network explicitly set:

Example

```
[basecm11->device[lite02]]% interfaces
[basecm11->device[lite02]->interfaces]% list
Type         Network device name  IP              Network          Start if
------------ -------------------- --------------- ---------------- --------
physical     eth0                 10.142.0.1      internalnet2     always
[basecm11->device[lite02]->interfaces]% ..
[basecm11->device[lite02]]% get managementnetwork

[basecm11->device[lite02]]% set managementnetwork
externalnet   globalnet     internalnet   internalnet2
[basecm11->device[lite02]]% set managementnetwork internalnet2
[basecm11->device*[lite02*]]% commit
```

After `register_node` has successfully finished its configuration in Linux, CMDaemon Lite runs as a service: so that the following commands work as expected:

```
[root@lite01 ~]# systemctl status cm-lite-daemon
...
[root@lite01 ~]# systemctl start cm-lite-daemon
...
[root@lite01 ~]# systemctl stop cm-lite-daemon
...
```

On non-Linux operating systems, `cm-lite-daemon` can be run on startup automatically. For example with `launchd` in macOS, or as a scheduled task in Microsoft Windows. It can be tested by first running it manually. If successful, the lite node should show up as being in the UP state in Base View or `cmsh`.

CMDaemon Lite On Cumulus Switches

From NVIDIA Base Command Manager version 10 onwards, Cumulus switches (section 3.9) also support running CMDaemon Lite. In that case, the property `hasclientdaemon` must be set for the switch.

Example

```
[head]% device add switch cumulus01
[head->device*[cumulus01*]]% # hasclientdaemon: set if CMDaemon Lite should run on switch
[head->device*[cumulus01*]]% set hasclientdaemon yes
[head->device*[cumulus01*]]% interfaces
[head->device*[cumulus01*]->interfaces]% add physical eth0
[head->device[cumulus01*]->interfaces*[eth0*]]% set mac 12:34:56:78:90:AB
[head->device[cumulus01*]->interfaces*[eth0*]]% set ip 1.2.3.4
[head->device[cumulus01*]->interfaces*[eth0*]]% commit
[head->device[cumulus01]->interfaces[eth0]]% ..;..
[head->device[cumulus01]]% status
cumulus01 ................ [  UP  ]
```

The ZTP settings should be configured from `ztpsettings` mode, and a username and password must be set within the `accesssettings` mode for the Cumulus switch. Further details on configuring Cumulus switches are given in section 3.9.

Even Lighter Than CMDaemon Lite: Configuring A Device As A Generic Node

To put things in perspective: so far the options described have been:
1. CMDaemon running on the device
2. CMDaemon Lite running on the device

A third option that can be considered, is to have
3. no CMDaemon at all running on the device and to register the device as a generic node with the regular CMDaemon on the head node. A generic node is a generic network device (page 117) that happens to be a node.

This third option—the generic node option—then monitors the device for a basic state of UP/DOWN, but nothing else. In contrast to the first two cases, a node license is not used up.

Even lighter than generic nodes: configuring a device as unmonitored: Devices can alternatively be added to the BIND DNS entries of the zone file via the `/var/named/*.include` files (Appendix A.1). This is a feature of the Linux operating system rather than a feature of BCM, and so —perhaps rather obviously—a BCM node license is also not used up in this case.

After restarting the named service, the nodes are not seen on the head node, and the device is not monitored in any way. The cluster does however know how to reach it, which in some cases may be all that a cluster administrator wants.

For example, if a host 10.141.1.20 with hostname `myotherhost01`, is added to the internalnet network within the domain name `eth.cluster`, then the session may be run as follows:

Example

```
[root@head ~]# vi /var/named/eth.cluster.zone.include
(appropriate DNS forward entry is added)
[root@head ~]# cat /var/named/eth.cluster.zone.include
myotherhost01 IN A 10.141.1.20
[root@head ~]# vi /var/named/141.10.in-addr.arpa.zone.include
(appropriate DNS reverse entry is added)
[root@head ~]# cat /var/named/141.10.in-addr.arpa.zone.include
20.1 IN PTR myotherhost01.eth.cluster.
[root@head ~]# systemctl restart named
```

3

Configuring The Cluster

After the NVIDIA Base Command Manager software has been installed on the head node, the cluster must be configured. For convenience, the regular nodes on the cluster use a default software image stored on the head node. The image is supplied to the regular nodes during a process called provisioning (Chapter 5), and both the head node and the regular nodes can have their software modified to suit exact requirements (Chapter 9). This chapter however goes through a number of basic cluster configuration aspects that are required to get all the hardware up and running on the head and regular nodes.

Section 3.1 explains how some of the main cluster configuration settings can be changed.

Section 3.2 details how the internal and external network parameters of the cluster can be changed.

Section 3.3 describes the setting up of network bridge interfaces.

Section 3.4 describes VLAN configuration.

Section 3.5 describes the setting up of network bond interfaces.

Section 3.6 covers how InfiniBand is set up.

Section 3.7 describes how Baseboard Management Controllers such as IPMI, iLO, DRAC, CIMC, and Redfish are set up.

Section 3.8 describes how switches are set up.

Section 3.9 describes how Cumulus switches are configured.

Section 3.11 describes how NetQ can be integrated with BCM.

Section 3.12 describes how UFM can be integrated with BCM.

Section 3.13 explains how disk layouts are configured, as well as how diskless nodes are set up.

Section 3.14 describes how NFS volumes are exported from an NFS server and mounted to nodes using the integrated approach of BCM.

Section 3.15 describes how services can be run from BCM.

Section 3.16 describes how a rack can be configured and managed with BCM.

Section 3.17 describes how GPUs can be configured with BCM.

Section 3.18 describes how sampling from a Prometheus exporter can be configured with BCM.

Section 3.19 describes how custom scripts can replace some of the default scripts in use.

Section 3.20 discusses configuration alternatives that are not based on CMDaemon.

Section 3.21 describes how the configuration files prior to a configuration change can be saved.

More elaborate aspects of cluster configuration such as power management, user management, package management, and workload management are covered in later chapters.

3.1 Main Cluster Configuration Settings

While both front ends—`cmsh` and Base View—can be used to carry out cluster management (Chapter 2), the BCM manuals often describe configuration with an arbitrary front end rather than for both front ends.

This is because the front ends are usually analogous enough to each other when carrying out a configuration procedure, so that describing the procedure for the other front end in detail as well is mostly wasteful. If the procedures differ significantly, then guidance is typically given on the differences.

Thus, for example, both `cmsh` and Base View can be used for the global configuration of cluster settings. For the `cmsh` front end, the configuration is done using `partition` mode. The analog to global configuration in the Base View front end relies on the navigation path: `Cluster` > `Settings`.

This section now continues with the Base View description.

The navigation path: `Cluster` > `Settings` brings up Base View's cluster settings window (figure 3.1):

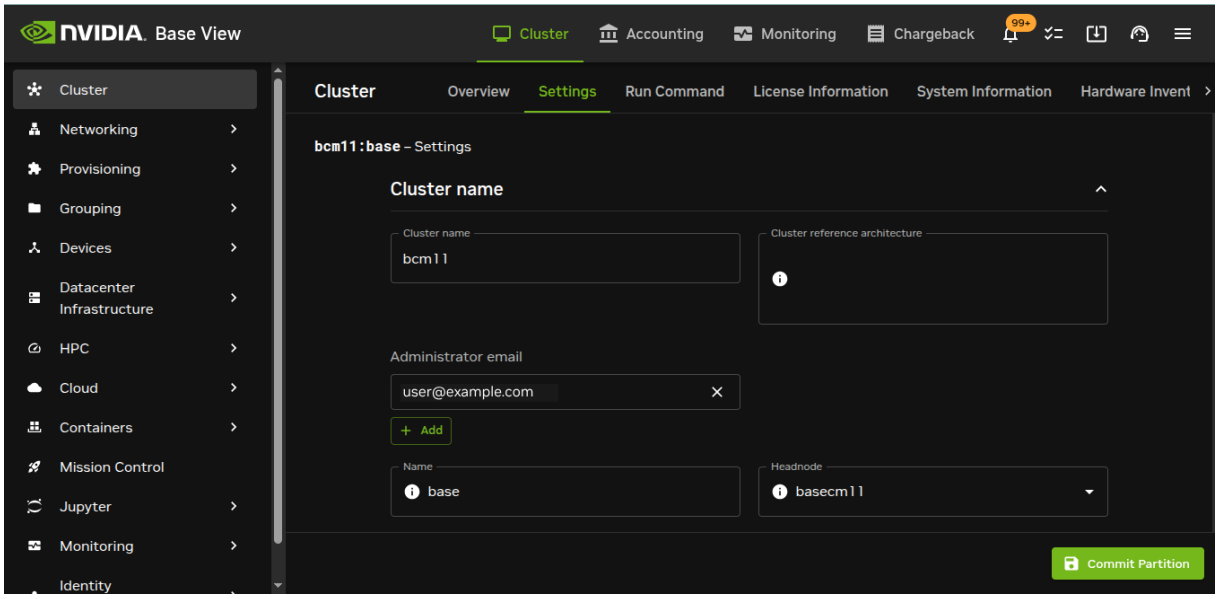


Figure 3.1: Cluster Settings

The navigational overview (page 41) indicates how the main cluster `Settings` window fits into the organizational layout of BCM.

The cluster `Settings` tab window allows changes to be made to many of the global cluster settings, typically using a grouping configuration that corresponds to the partition-level configuration in `cmsh`. The values that can be set for these global settings can in some cases be overridden by more specific configuration levels, such as (configuration-)overlay-level, category-level, or node-level configuration. The main cluster settings in figure 3.1 are about name-related settings, cluster networking settings, and some other miscellaneous global cluster settings.

3.1.1 Cluster Configuration: Various Cluster-wide Settings

In the Base View cluster `Settings` tab window of figure 3.1, the following cluster-wide settings can be viewed and modified:

- `Cluster name`: Default value: `BCM HEAD Cluster`.
- `Administrator email`: Default value: Specified during installation (figure 3.8 in Chapter 3 of the *Installation Manual*).
- `Name servers`: No default value, or picked up using DHCP.
- `Time servers`: Default values: `0.pool.ntp.org`, `1.pool.ntp.org`, `2.pool.ntp.org`.
- `Search domains`: No default value.
- `Externally visible IP`: Default value: `0.0.0.0`.

- `Time zone`: Default value: Specified during installation (figure 3.8 in Chapter 3 of the *Installation Manual*).
- `BMC settings`: This allows BMC and related settings to be managed.

  The following objects already exist by default on a new cluster, and can have their settings modified:

  - BMC (section 3.1.3)
  - SNMP (section 3.8.2)
  - Provisioning (Chapter 5, section 5.2.4)

  The following objects do not already exist by default on a new cluster, which means that clicking on them to configure them begins the process of creating the object, prompting the administrator to enter values for the object:

  - SELinux (Chapter 12 of the *Installation Manual*)
  - Access (section 3.9.1)
  - ZTP (section 3.9)
  - ZTP new switch
  - NetQ (section 3.11)
  - UFM (section 3.12)
  - NMX Manager (section 2.3 of the *NVIDIA Mission Control Manual*)

  The BMC and related settings are described in their associated sections.
- `External network`: Default value: as defined during selection of network topology during installation. The `Management network` can also be configured in this section.
  - `Management network`: Default value: as defined during selection of network topology during installation.

  Network topology selection during installation is covered in section 3.3.9 of the *Installation Manual*.
- `Default category`: Default value: `default`. This value is inherited as the default object name for `category` mode.
- `ArchOS`: No default value. This specifies an architecture and OS.
- `Sign installer certificates`: Default value: `AUTO`. Described in section 5.4.1.
- `Failover`: No default value.
- `Failover groups`: No default value.
- `Default burn configuration`: Default value: `default-destructive`.
- `Wlm job power usage settings`: No default value.
- `Leak action policies`: Default value: `Default building policy`.
- `Building Management System`: Default value: `NVIDIA-conforming BMS`.
- `Prometheus metric forwarders`: No default value.

3.1.2 Cluster Configuration: Some Network-Related Settings

The following network-related settings are also described in the context of external network settings for the cluster object, in section 3.2.3, as well as in the quickstart in Chapter 1 of the *Installation Manual*.

Nameserver And Search Domains Used By Cluster

- If Base View is used, then the settings window can be used to set the IP address of the nameserver and the names of the search domains for the cluster.

  By default, the nameserver is the internal (regular-nodes-facing) IP address of the head node. Multiple nameservers can be added. If the value is set to `0.0.0.0`, then the address supplied via DHCP to the head node is used. Alternatively, a static value can be set. Static IP addresses must be used for external addresses in the case of the cluster being configured with high availability.

- If `cmsh` is used instead of Base View, then the changes to the `nameserver` and `searchdomain` values can instead be carried out via `partition` mode (page 113).

Limit to the number of search domains: In older versions of glibc (before 2.26), the number of names that can be set as search domains used by the cluster has a hardcoded maximum limit of 6, with a total of 256 characters.

From glibc 2.26 onward, this operating system limit no longer applies to normal lookups. BCM applies its own limit instead, using the `MaximalSearchDomains` `GlobalConfig` directive (Appendix C, page 917), which defaults to 64 search domains, and which can be set to any value from 1 to 1024.

Raising `MaximalSearchDomains` above 6 only has an effect on a cluster running glibc 2.26 or later. On an older glibc, only the first 6 configured search domains are used, regardless of the directive's setting.

Changing The Order In `resolv.conf`

For clusters, CMDaemon by default automatically writes the `/etc/resolv.conf` by using the following sources, and in the following order:

1. Global network
2. Other networks
3. Category search domains
4. Partition search domains

Because older glibc versions only support 6 entries in `/etc/resolv.conf`, it is sometimes useful to exclude or reorder the preceding sources.

For a network object, there are two fields that control the position of the domain name in the file `/etc/resolv.conf`:

Example

```
[basecm11]% network use ibnet
[basecm11->network[ibnet]]% show
...
Exclude from search domain       no
Search domain index              0
```

If the `Exclude from search domain` field is set to `yes`, then the domain name for the network is not used.

The `Search domain index` field specifies the position of the domain name. A value of `0` means CMDaemon automatically determines its location.

The index of the category and partition search domains can also be changed by appending a number, in the suffix format `:<index>`, to the domain name:

Example

```
[basecm11]% partition
[basecm11->partition[base]]% get searchdomains
example.com:1
domain.test:6
```

If an index is set for one search domain, then setting indices for all search domains is recommended. Search domains without indices are handled automatically by CMDaemon.

CMDaemon sorts all search domains according to index, and writes `/etc/resolv.conf` with the 6 that have the lowest index, with the lowest index first.

Setting The Stub Resolver For Ubuntu Hosts

For Ubuntu, CMDaemon generates a `/etc/resolv.conf` file (Appendix A), and symlinks it to the `resolv.conf` managed by `systemd` at `/etc/run/systemd/resolve/resolve.conf`.

However, a default Ubuntu without BCM uses a systemd stub resolver service (`man systemd-resolved.8`) that listens on the IP address 127.0.0.53. Sometimes a stub resolver is also useful in BCM for containers such as enroot containers (section 7.3.3) that mount and use `/etc/resolv.conf`.

To enable resolution for such containers on a node via the stub resolver instead of the uplink resolver, the resolver can be set using the extra `resolve` parameter, using the `-e|--extra` setting:

Example

```
basecm11->device[node001]]% # to set it to the stub resolver:
basecm11->device[node001]]% set -e resolv stub
basecm11->device*[node001*]]% commit        # wait a bit
basecm11->device[node001]]% # to set it back to the BCM value:
basecm11->device[node001]]% set -e resolv uplink
basecm11->device*[node001*]]% commit       # wait a bit
```

Externally Visible IP Address

The externally visible IP address are public (non-RFC 1918) IP addresses to the cluster. These can be set to be visible on the external network.

- If using Base View, the navigation path is:
  `Cluster` > `Settings` > `Externally visible IP`.
- For `cmsh`, the parameter `externallyvisibleip` can be set via `partition` mode.

Time server(s)

Time server hostnames can be specified for the NTP client on the head node.

- If using Base View, the navigation path is via `Cluster` > `Settings` > `NAME SERVERS`.
- For `cmsh`, the parameter `timeservers` can be set via `partition` mode.

Time Zone

The time zone setting can be set at various grouping levels:

If applied to the entire cluster, and if it is applied in `partition` mode, then:

- In Base View, the time zone parameters can be jumped to via the navigation path:
  `Cluster` > `Settings` > `Time zone`
- In `cmsh`, the time zone can be selected in `partition` mode, using the `base` object. Tab-completion prompting after entering "`set timezone`" displays a list of several hundred possible time zones, from which one can be chosen:

Example

```
[basecm11]% partition use base
[basecm11->partition[base]]% set timezone america/los_angeles
[basecm11->partition*[base*]]% commit
```

A time zone setting can also be applied at the level of a node, category, edge site, and cloud region. As is usual in the BCM hierarchy, the value set for the larger grouping is the default value used by the members of that group, while a value set specifically for the individual members of that group overrides such a default.

3.1.3 Miscellaneous Settings

BMC (IPMI/iLO, DRAC, CIMC, Redfish) Settings

The BMC (Baseboard Management Controller) access settings can be configured in Base View via the navigation path:

`Cluster > Settings > BMC Settings >`

This opens up a window so that the BMC settings can be managed:

- `User name`: (default: `bright`)
- `Password`: (default: random string generated during head node installation)
- `User ID`: (default: `4`)
- `Power reset delay`: During a reset, this is the time, in seconds, that the machine is off, before it starts powering up again (default: `0`)
- `Extra arguments`: (default: none)
- `privilege`: (default: `administrator`)

The defaults in the preceding are set when the BMC interfaces are configured during head node installation. If the BMC interfaces are not set then, then the defaults are also unset.

BMC configuration is discussed in more detail in section 3.7.

Administrator E-mail Setting

By default, the distribution which BCM runs on sends e-mails for the administrator to the root e-mail address. The administrator e-mail address can be changed within BCM so that these e-mails are received elsewhere.

- In Base View, an e-mail address (or space-separated addresses) can be set in the `Administrator e-mail` field via the navigation path `Cluster > Settings > Administrator e-mail`
- In `cmsh`, the e-mail address (or space-separated addresses) can be set in `partition` mode, using the `base` object as follows:

Example

```
[basecm11]% partition use base
[basecm11->partition[base]]% set administratore-mail alf@example.com beth@example.com
[basecm11->partition*[base*]]% commit
```

The following critical states or errors cause e-mails to be sent to the e-mail address:

- By default, a month before the cluster license expires, a reminder e-mail is sent to the administrator account by CMDaemon. A daily reminder is sent if the expiry is due within a week.
- A service on the head node that fails on its first ever boot.
- When an automatic failover fails on the head or regular node.

SMTP Relay Host Mailserver Setting

The head node uses Postfix as its SMTP server. The default base distribution configuration is a minimal Postfix installation, and so has no value set for the SMTP relay host. To set its value:

- in Base View: the `Relay Host` field, with the navigation path:

  `Cluster > Settings > Search domains > Relay Host`

  sets the SMTP relay host for the cluster resource.

- in `cmsh`: the `relayhost` property can be set for the `base` object within `partition` mode:

  Example

```
[root@basecm11 ~]# cmsh
[basecm11]% partition use base
[basecm11-> partition[base]]% set relayhost mail.example.com
[basecm11-> partition[base*]]% commit
```

Postfix on the regular nodes is configured to use the head node as a relay host and is normally left untouched by the administrator.

If the regular node configuration for Postfix is changed due to a `partition` mode change, then a node reboot deploys the change for the node. Setting the `AdvancedConfig` (page 925) `SmtpPartitionRelayHostInImages` to `0` disables the changing of the relay host on the regular node.

Further Postfix changes can be done directly to the configuration files as is done in the standard distribution. The changes must be done after the marked auto-generated sections, and should not conflict with the auto-generated sections.

A convenient way to check mail is functioning is to run BCM's `testemail` command. The command is run from within the `main` mode of `cmsh`. It sends a test e-mail out using CMDaemon:

```
[root@basecm11 ~]# mailq; ls -al /var/spool/mail/root
Mail queue is empty
-rw------- 1 root mail 0 Sep  8 11:11 /var/spool/mail/root
[root@basecm11 ~]# cmsh -c "main; testemail"
Mail delivered to postfix
You have new mail in /var/spool/mail/root
[root@basecm11 ~]# mailq; ls -al /var/spool/mail/root
Mail queue is empty
-rw------- 1 root mail 749 Sep  8 11:12 /var/spool/mail/root
```

The test e-mail destination is the administrator e-mail address discussed in the preceding section.

Failover Settings

To access options to the high availability (HA) feature of the cluster for head nodes, the administrator can click on the `Failover` option in Base View. This opens up a subwindow that can be used to configure HA-related parameters (section 15.4.6).

Failover Groups Settings

To access options to the high availability feature of the cluster for groups of regular nodes, the administrator can click on the `Failover groups` option in Base View. This opens up a subwindow that can be used to configure failover-groups-related parameters (section 15.5).

Burn Configs

Burning nodes is covered in Chapter 11 of the *Installation Manual*. Burn configuration settings for the cluster can be accessed in Base View via the following navigation paths:

- `Cluster > Settings > Default burn configuration`

  This allows the `Default burn configuration` for a node burn run to be modified.

- `Cluster > Settings > Burn configs`

  This lists the possible burn configuration settings in a subwindow, and allows changes to some of their properties of each item of the list.

FIPS Mode

To be compliant with the Federal Information Processing Standards of the USA, Linux-based systems are required to follow some security standards. This is known as FIPS compliance.

FIPS compliance on regular nodes can be set per node or per category.

The existing FIPS status can be checked with the `sysinfo` command at node level:

Example

```
[head->device]% sysinfo node002
Name                      Value
------------------------- --------------------------------
...
FIPS                      No
```

Or even just with a `get`:

Example

```
[head->device]% get node002 fips
no
```

FIPS can be enabled per node:

Example

```
[head->device]% set node002 fips yes
[head->device*]% commit
[head->device]% reboot node002
        [time passes]
        [node002 finishes reboot]
[head->device]% sysinfo node002 | grep FIPS
FIPS                      Yes
```

FIPS can be enabled per category:

Example

```
[head->device]% sysinfo node002
[head->device]% category
[head->category]% set default fips yes
[head->category*]% commit
Successfully committed 1 Categories
[head->category]%
...[notice] head: node001 [   UP   ], restart required (fips)
...[notice] head: node002 [   UP   ], restart required (fips)
[head->category]% device reboot -n node001..node002
[head->category]% device
[head->device]% get node001 fips; get node002 fips
yes
yes
```

As is usual in BCM, the node-level setting overwrites the category-level setting.

Setting FIPS compliance on the head node itself via CMDaemon is not allowed. This is because FIPS GRUB configuration could, in some unusual cases, result in an unbootable head node. The cluster administrator must consider the existing state of the head node with due care before manually reconfiguring it for FIPS.

3.1.4 Limiting The Maximum Number Of Open Files

Configuring The System Limit On Open Files: The `/proc/sys/fs/file-max` Setting

The maximum number of open files allowed on a running Linux operating system is determined by `/proc/sys/fs/file-max`. To configure this setting so that it is persistent, the Linux operating system uses a `/etc/sysctl.conf` file and `*.conf` files under `/etc/sysctl.d/`. Further information on these files can be found via the man page, `man sysctl.conf.5`. BCM adheres to this standard method, and places a settings file `90-cm-sysctl.conf` in the directory `/etc/sysctl.d`.

By default, the value set for `file-max` by NVIDIA Base Command Manager is 131072. An on-premises head node typically is not used to run applications that exceed this value, and the settings file `90-cm-sysctl.conf` is therefore not available for modification by the CMDaemon front ends (`cmsh` and Base View). For edge and cloud director installations, the installation scripts take care of modifying `file-max` since such installations can have many more open files. Cluster administrators are thus not expected to have a need to modify the value of `file-max` in most use cases.

However, if there is a need to modify the value of `file-max`, then a subsequent, extra, configuration file, such as `/etc/sysctl.d/91-site-sysctl-file-max-tweak.conf` can be added under `/etc/sysctl.d/`, so that the BCM version is not intefered with directly.

Configuring The User Limit On Open Files: The `nofile` Setting

The maximum number of open files allowed for a user can be seen on running `ulimit -n`. The value is defined by the `nofile` parameter of ulimit.

By default the value set by BCM is 131072.

Ulimit limits are limits to restrict the resources used by users. If the `pam_limits.so` module is used to apply ulimit limits, then the resource limits can be set via the `/etc/security/limits.conf` file and `*.conf` files in the `/etc/security/limits.d` directory. Further information on these files can be found via the man page, `man limits.conf.5`.

Resource limits that can be set for user login sessions include the number of simultaneous login sessions, the number of open files, and memory limits.

The maximum number of open files for a user is unlimited by default for an operating system that is not managed by BCM. However, it is set to 131072 by default for a system managed by BCM. The `nofile` value is defined by BCM in:

- in `/etc/security/limits.d/91-cm-limits.conf` on the head node
- in `/cm/images/<`*software image name*`>/etc/security/limits.d/91-cm-limits.conf` in the software image that the regular node picks up.

The values set in `91-cm-limits.conf` are typically sufficient for a user session, unless the user runs applications that are resource hungry and consume a lot of open files.

Deciding On Appropriate Ulimit, Limit, And System Limit Values

Decreasing the `nofile` value in `/etc/security/limits.d/91-cm-limits.conf` (but leaving the `/proc/sys/fs/file-max` untouched), or increasing `/proc/sys/fs/file-max` (but leaving the `nofile` value of 131072 per session as is), may help the system stay under the maximum number of open files allowed.

In general, users should not be allowed to use the head node as a compilation server, or as a test bed, before running their applications. This is because user errors can unintentionally cause the head node to run out of resources and crash it.

Depending what is running on the server, and the load on it, the administrator may wish to increase the resource limit values.

A very rough rule-of-thumb that may be useful as a first approximation to set `file-max` optimally is suggested in the kernel source code. The suggestion is to simply multiply the system memory (in MB) by 10 per MB, and make the resulting number the `file-max` value. For example, if the node has 128 GB of memory, then 1280000 can be set as the `file-max` value.

Fine-tuning to try and ensure that the operating system no longer runs out of file handles, and to try and ensure the memory limits for handling the load are not exceeded, is best achieved via an empirical trial-and-error approach.

3.2 Network Settings

A simplified quickstart guide to setting up the external head node network configuration on a vendor-prepared cluster is given in Chapter 6 of the *Installation Manual*. This section (3.2) covers network configuration more thoroughly.

After the cluster is set up with the correct license as explained in Chapter 4 of the *Installation Manual*, the next configuration step is to define the networks that are present (sections 3.2.1 and 3.2.2).

During BCM installation at least three default network objects were created:

`internalnet`: the primary internal cluster network, and the default management network. This is used for booting non-head nodes and for all cluster management communication. In the absence of other internal networks, `internalnet` is also used for storage and communication between compute jobs. Changing the configuration of this network is described on page 114 under the subheading "Changing Internal Network Parameters For The Cluster".

`externalnet`: the network connecting the cluster to the outside world (typically a corporate or campus network). Changing the configuration of this network is described on page 110 under the subheading "Changing External Network Parameters For The Cluster". This is the only network for which BCM also supports IPv6.

`globalnet`: the special network used to set the domain name for nodes so that they can be resolved whether they are cloud nodes or not. This network is described further on page 117 under the subheading "Changing The Global Network Parameters For The Cluster".

For a Type 1 cluster (section 3.3.9 of the *Installation Manual*) the internal and external networks are illustrated conceptually by figure 3.2.

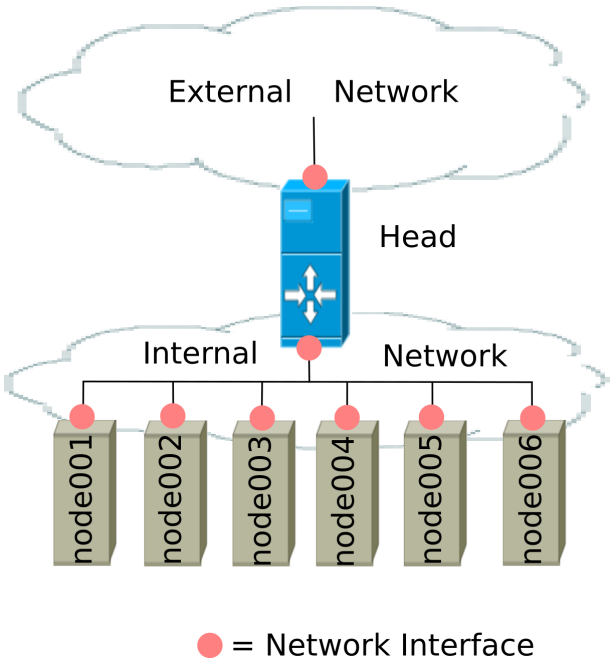


Figure 3.2: Network Settings Concepts

The configuration of network settings is completed when, after having configured the general network settings, specific IP addresses are then also assigned to the interfaces of devices connected to the networks.

- Changing the configuration of the head node external interface is described on page 111 under the subheading "The IP address of the cluster".
- Changing the configuration of the internal network interfaces is described on page 115 under the subheading "The IP addresses and other interface values of the internal network".
  - How to set a persistent identity for an interface—for example, to ensure that a particular interface that has been assigned the name `eth3` by the kernel keeps this name across reboots—is covered in section 5.8.1, page 311.
- Changing the configuration of `globalnet` is described on page 117 under the subheading "Changing The Global Network Parameters For The Cluster". IP addresses are not configured at the `globalnet` network level itself.

3.2.1 Configuring Networks

The `network` mode in `cmsh` gives access to all network-related operations using the standard object commands. Section 2.5.3 introduces `cmsh` modes and working with objects.

In Base View, a network can be configured via the navigation path `Networking` > `Networks`, which opens up the `Network list` subwindow (figure 3.3):



Figure 3.3: Networks

In the context of the OSI Reference Model, each network object represents a layer 3 (i.e. Network Layer) IP network, and several layer 3 networks can be layered on a single layer 2 network (e.g. routes on an Ethernet segment).

Opening up the subwindow for a network such as `internalnet` or `externalnet` brings up the `Devices Overview` tab window, which lists the devices on the network and associated properties. For example, the `internal` network typically has some nodes and other devices attached to it, each with their IP addresses and statuses (figure 3.4).

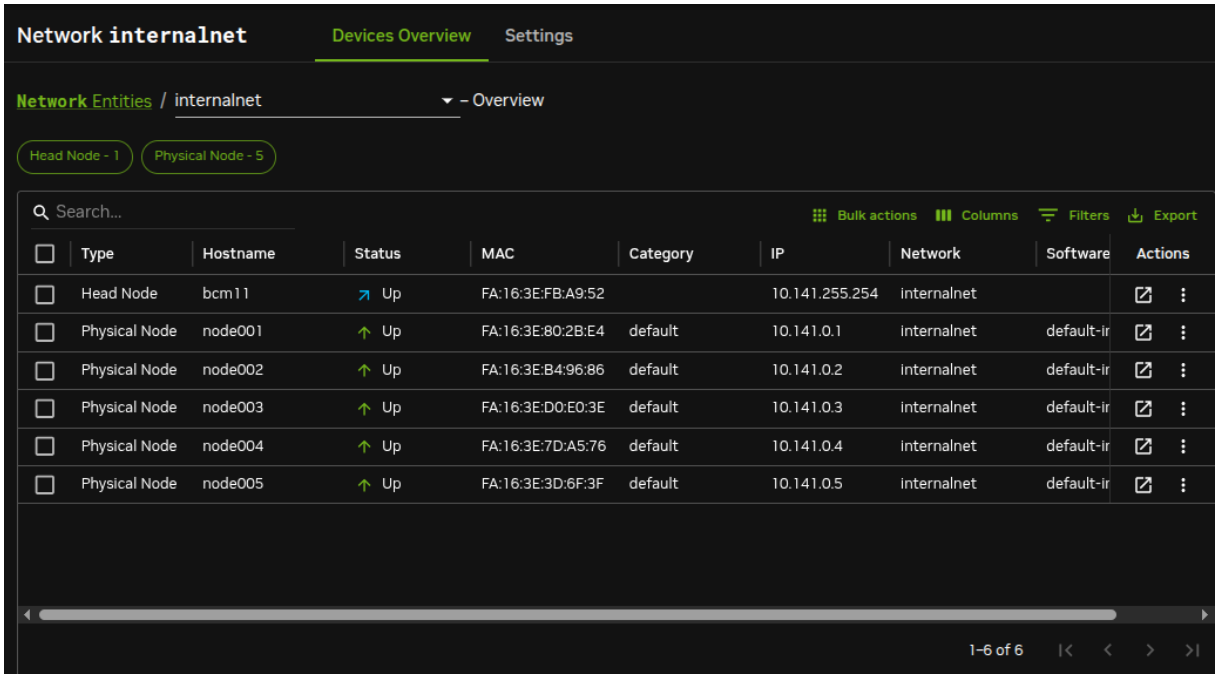


Figure 3.4: Network Overview

Selecting the `Settings` tab window opens a scrollable pane that allows a number of network prop-

erties to be changed (figure 3.5).

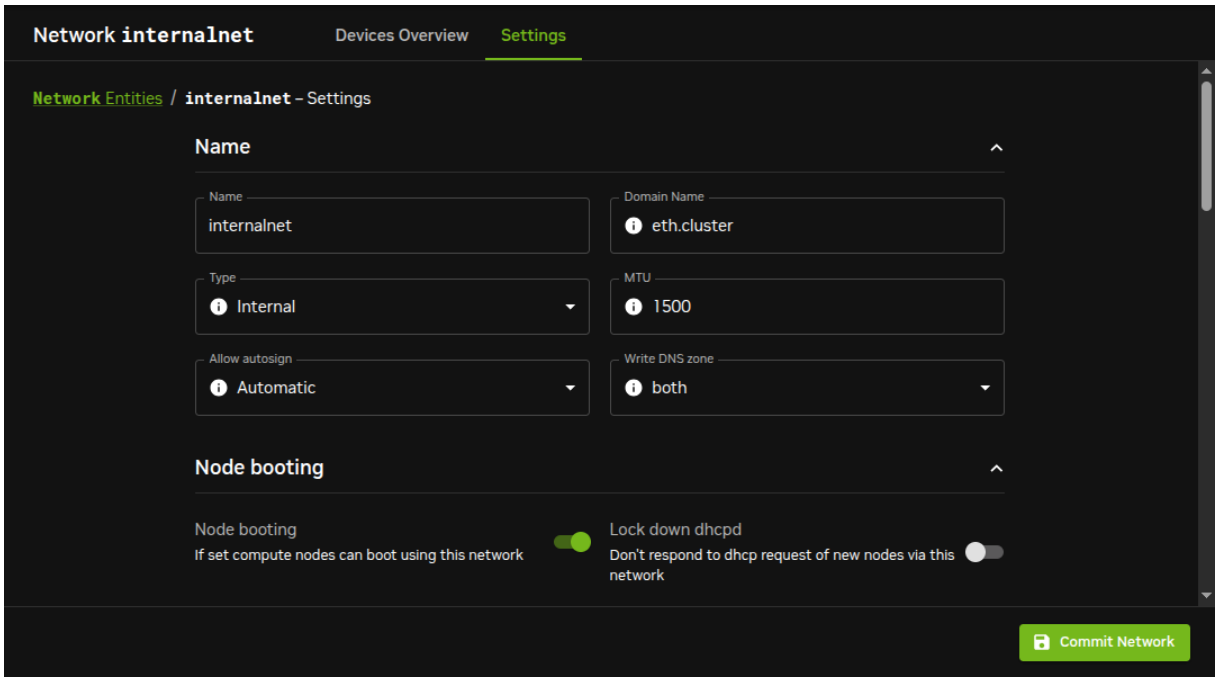


Figure 3.5: Network Settings

The properties of figure 3.5 are introduced in table 3.1:

Property	Description
`Name`	Name of this network.
`Domain Name`	DNS domain associated with the network.
`Type`	Menu options to set the network type. Options are `Internal`, `External`, `Tunnel`, `Global`, `Cloud`, or `NetMap` network.
`MTU`	Maximum Transmission Unit. The maximum size of an IP packet transmitted without fragmenting.
`Allow autosign`	Set if node-installer certificate sign requests are signed automatically.
`Write DNS zone`	Set which DNS zones should be written (forward or reverse).
`Node booting`	Enabling means nodes are set to boot from this network (useful in the case of nodes on multiple networks). For an internal subnet called <*subnet*>, when node booting is set, CMDaemon adds a subnet configuration `/etc/dhcpd.`<*subnet*>`.conf` on the head node, which is accessed from `/etc/dhcpd.conf`. • It can be set in Base View using the navigation path `Networking` > `Networks` > `internalnet` > > `Settings` > `Node booting`. • It can be set in `cmsh` using the `network` mode, selecting the network for the node, and then setting `nodebooting`.

...continues

...continued

Property	Description
Lock down dhcpd	Enabling means new nodes are not offered a PXE DHCP IP address from this network, i.e. DHCPD is "locked down". A DHCP "deny unknown-clients" option is set by this action, so no new DHCP leases are granted to unknown clients for the network. Unknown clients are nodes for which BCM has no MAC addresses associated with the node. • It can be set in Base View via the Networking resource, selecting a network, and then setting Lock down dhcpd from within the network settings. • It can be set in cmsh via the network mode, selecting a network, and then setting lockdowndhcpd.
IMEX	Enabling means the network is used for the NVIDIA IMEX peer list. The fully qualified domain name (FQDN) of each node on this network is then written to /etc/nvidia-imex/nodes_config.cfg instead of the short hostname (section 7.5.1 for the resulting configuration file and further IMEX details). • It can be set in cmsh using the network mode, selecting the network, and then setting imex.
IMEX IP	Enabling means the IMEX peer list in nodes_config.cfg is written using the IP address of each node on this network rather than the FQDN. Requires IMEX to also be enabled on the network (section 7.5.1). • It can be set in cmsh using the network mode, selecting the network, and then setting imex ip.
Management allowed	Enabling means that the network has nodes managed by the head node.
Search domain index	The position of the network domain in the resolv.conf file. A value of 0 means the position is determined automatically.
Exclude from search domain	Enabling means the domain name for the network is not used.
Disable automatic exports	Enabling means that exports of built-in filesystems are not done automatically for the network.
Base address	Base address of the network (also known as the *network address*).
Dynamic range start/end	Start/end IP addresses of the DHCP range temporarily used by nodes during PXE boot on the internal network. These are addresses that do not conflict with the addresses assigned and used by nodes during normal use.
Netmask bits	Prefix-length, or number of bits in netmask. The part after the "/" in CIDR notation.
Gateway	Default route IP address
Gateway metric	Gateway metric
IPv6 enabled	Enables IPv6
IPv6 gateway	Sets IPv6 gateway
IPv6 base address	Sets IPv6 base address
IPv6 netmask bits	Sets IPv6 netmask bits
Cloud Subnet ID	Cloud ID of subnet
EC2AvailabilityZone	AWS availability zone in which the subnet exists
Layer3	If set to yes it creates layer 3 connections with /31 netmasks. The following parameters can then be set: • Layer3 route: Sets a layer 3 route • Layer3 ecmp: Creates a layer 3 network with equal-cost multi-path

...continued

Property	Description
Notes	Notes can be added for the network

Table 3.1: *Network Configuration Settings*

Basic Networking Concepts And The `sipcalc` Utility

In basic networking concepts, a network is a range of IP addresses. The first address in the range is the *base address*. The length of the range, i.e. the *subnet*, is determined by the *netmask*, which uses CIDR notation. CIDR notation is the so-called / ("slash") representation, in which, for example, a CIDR notation of 192.168.0.1/28 implies an IP address of 192.168.0.1 with a traditional netmask of 255.255.255.240 applied to the 192.168.0.0 network. The netmask 255.255.255.240 implies that bits 28–32 of the 32-bit dotted-quad number 255.255.255.255 are unmasked, thereby implying a 4-bit-sized host range of 16 (i.e. 2^4) addresses.

The `sipcalc` utility can be installed on the head node, and is a useful tool for calculating or checking such IP subnet values (`man sipcalc.1` or `sipcalc -h` for help on this utility):

Example

```
user@basecm11:~$ sipcalc 192.168.0.1/28
-[ipv4 : 192.168.0.1/28] - 0

[CIDR]
Host address            - 192.168.0.1
Host address (decimal)  - 3232235521
Host address (hex)      - C0A80001
Network address         - 192.168.0.0
Network mask            - 255.255.255.240
Network mask (bits)     - 28
Network mask (hex)      - FFFFFFF0
Broadcast address       - 192.168.0.15
Cisco wildcard          - 0.0.0.15
Addresses in network    - 16
Network range           - 192.168.0.0 - 192.168.0.15
Usable range            - 192.168.0.1 - 192.168.0.14
```

DNS Domains And Zones

Every network has an associated DNS domain which can be used to access a device through a particular network. For `internalnet`, the default DNS domain is set to `eth.cluster`, which means that the hostname `node001.eth.cluster` can be used to access device `node001` through the primary internal network. If a dedicated storage network has been added with DNS domain `storage.cluster`, then `node001.storage.cluster` can be used to reach `node001` through the storage network. Internal DNS zones are generated automatically based on the network definitions and the defined nodes on these networks. For networks marked as external, no DNS zones are generated.

Layer 3 Properties

Setting the layer 3 properties of table 3.1 with `set layer3 yes` configures the nodes in the (main) network range to be routed by a switch with layer 3 features. The default netmask when `layer3` is set to `yes` is /31 (255.255.255.254) for each device and switch port pair in the subnets that are created by this setting, while routing stays defined by the netmask of the (main) network, and the routing is handled by the layer 3 features of the switch.

The netmask for layer 3 networking can be set to /30 with `set -e layer3bits 30`. This is needed for some DataDirect Networks switches.

3.2.2 Adding Networks

Once a network has been added, it can be used in the configuration of network interfaces for devices.

In Base View the Add button in the Networks subwindow (figure 3.3) can be used to add a new network. After the new network has been added, the Settings pane (figure 3.5) can be used to further configure the newly-added network.

In cmsh, a new network can be added from within network mode using the add or clone commands.

The default assignment of networks (internalnet to Management network and externalnet to External network) can be changed using Base View, via the Settings window of the cluster object, which uses the navigation path:

Cluster > Settings > External network > {External network|Management network}

In cmsh the assignment to Management network and External network can be set or modified from the base object in partition mode:

Example

```
[root@basecm11 ~]# cmsh
[basecm11]% partition use base
[basecm11->partition[base]]% set managementnetwork internalnet; commit
[basecm11->partition[base]]% set externalnetwork externalnet; commit
```

3.2.3 Changing Network Parameters

After both internal and external networks are defined, it may be necessary to change network-related parameters from their original or default installation settings.

Changing The Head Or Regular Node Hostname

To reach the head node from inside the cluster, the alias master may be used at all times. Setting the hostname of the head node itself to master is not recommended.

The name of a cluster is sometimes used as the hostname of the head node. The cluster name, the head node hostname, and the regular node hostnames, all have their own separate names as a property of their corresponding objects. The name can be changed in a similar manner for each.

For example, to change the hostname of the head node, the device object corresponding to the head node must be modified.

- In Base View, the navigation path for modifying the host name is Devices > All Devices > Head Node >  > Settings > Hostname > Commit HeadNode. That is, under the Devices resource, the All Devices option is selected. The head node is selected from the devices displayed, and its subwindow-opening box in the Actions column is then clicked. This opens the device overview pane. The Settings option can then be selected. The Hostname record can then be modified (figure 3.6), and saved by clicking on the Commit HeadNode icon. When setting a hostname, a domain is not included.

  After the change, as suggested by Base View, the head node must be rebooted.

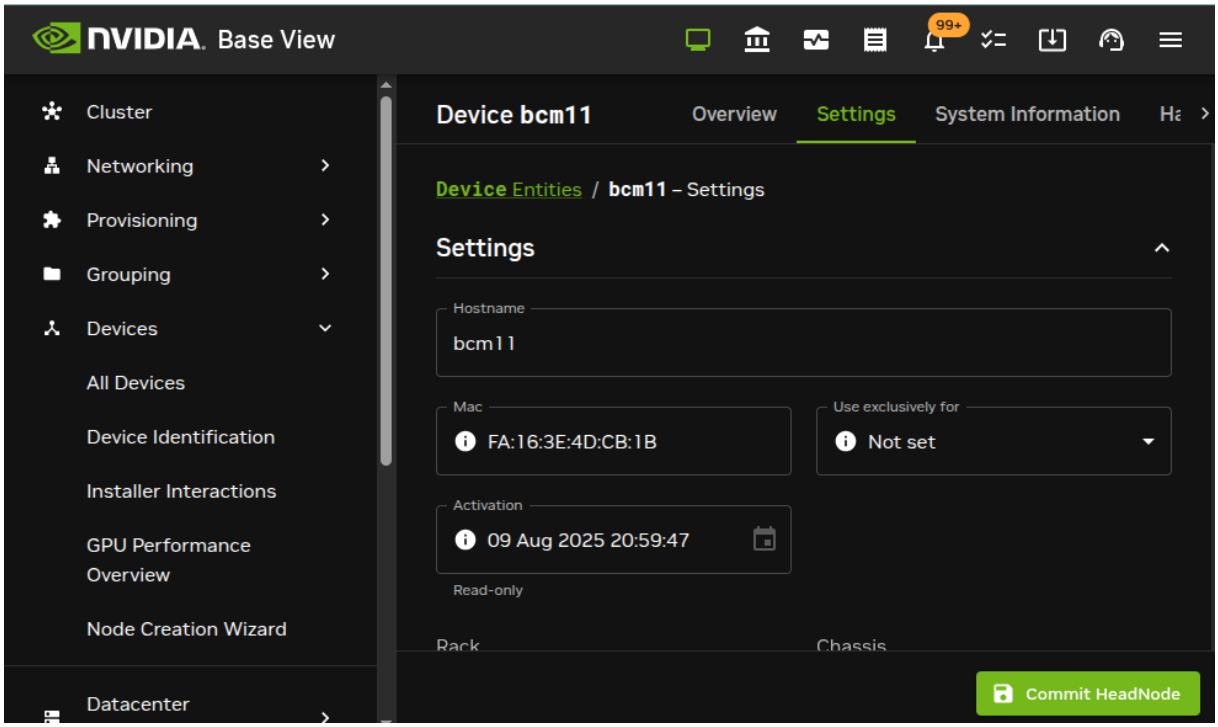


Figure 3.6: Head Node Settings

- In `cmsh`, the hostname of the head node can also be changed, via device mode:

  Example

```
[root@basecm11 ~]# cmsh
[basecm11]% device use basecm11
[basecm11->device[basecm11]]% set hostname foobar
[foobar->device*[foobar*]]% commit
[foobar->device[foobar]]%
Tue Jan 22 17:35:29 2013 [warning] foobar: Reboot required: Hostname changed
[foobar->device[foobar]]% quit
[root@basecm11 ~]# sleep 30; hostname -f foobar.cm.cluster
[root@basecm11 ~]#
```

  The prompt string shows the new hostname after a short time, when a new shell is started.

  After the hostname has been set, as suggested by `cmsh`, the head node must be rebooted.

Adding Hostnames To The Internal Network

Additional hostnames, whether they belong to the cluster nodes or not, can be added as name/value pairs to the `/etc/hosts` file(s) within the cluster. This should be done only outside the specially-marked CMDaemon-managed section. It can be done to the file on the head node, or to the file on the software image for the regular nodes, or to both, depending on where the resolution is required.

However, for hosts that are on the internal network of the cluster, such as regular nodes, it is easier and wiser to avoid adding additional hostnames via `/etc/hosts`.

Instead, it is recommended to let BCM manage host name resolution for devices on the `internalnet` through its DNS server on the `internalnet` interface. The host names can be added to the `additionalhostnames` object, from within `interfaces` submode for the head node. The `interfaces` submode is accessible from the `device` mode. Thus, for the head node, with `eth1` as the interface for `internalnet`:

Example

```
[basecm11]% device use basecm11
[basecm11->device[basecm11]]% interfaces
[basecm11->device[basecm11]->interfaces]% use eth1
[basecm11->device[basecm11]->interfaces[eth1]]% set additionalhostnames test
[basecm11->device*[basecm11*]->interfaces*[eth1*]]% commit
[basecm11->device[basecm11]->interfaces[eth1]]%
Fri Oct 12 16:48:47 2012 [notice] basecm11: Service named was restarted
[basecm11->device[basecm11]->interfaces[eth1]]% !ping test
PING test.cm.cluster (10.141.255.254) 56(84) bytes of data.
...
```

Multiple hostnames can be added as space-separated entries.

The named service automatically restarts within about 20 seconds after committal, implementing the configuration changes. This kind of restart is a feature (section 3.15.1) of changes made to service configurations by Base View or cmsh.

Changing External Network Parameters For The Cluster

The external network parameters of the cluster: When a cluster interacts with an external network, such as a company or a university network, its connection behavior is determined by the settings of two objects: firstly, the external network settings of the Networks resource, and secondly, by the cluster network settings.

1. **The external network object** contains the network settings for all objects configured to connect to the external network, for example, a head node. Network settings are configured in Base View under the Networking resource, then under the Networks subwindow, then within the Settings option for a selected network. Figure 3.5 shows a settings window for when the internalnet item has been selected, but in the current case the externalnet item must be selected instead. The following parameters can then be configured:

   - the IP network parameters of the cluster (but not the IP address of the cluster):
     - Base address: the network address of the external network (the "IP address of the external network"). This is not to be confused with the IP address of the cluster, which is described shortly after this.
     - Broadcast address: the broadcast address of the external network. This should not be confused with the broadcast IP address of the internal network.
     - Dynamic range start and Dynamic range end: Not used by the external network configuration.
     - Netmask bits: the netmask size, or prefix-length, of the external network, in bits.
     - Gateway: the default route for the external network.
   - the network name (what the external network itself is called), by default this is defined as externalnet in the base partition on a newly installed Type 1 cluster,
   - the Domain Name: the network domain (LAN domain, i.e. what domain machines on the external network use as their domain),
   - the External network checkbox: this is checked for a Type 1 cluster,
   - and MTU size (the maximum value for a TCP/IP packet before it fragments on the external network—the default value is 1500).

2. **The cluster object** contains other network settings used to connect to the outside. These are configured in Base View using the navigation path:

   Cluster > Settings

   and are described in section 3.1.1:

- e-mail address(es) for the cluster administrator,
- any additional external name servers used by the cluster to resolve external host names. As an aside: by default, only the head node name server is configured, and by default it only serves hosts on the internal network via the internal interface. Enabling the `PublicDNS` directive (Appendix C) allows the head node name server to resolve queries about internal hosts from external hosts via any interface, including the external interface.
- the DNS search domain (what the cluster uses as its domain),
- and NTP time servers (used to synchronize the time on the cluster with standard time) and time zone settings.

These settings can also be adjusted in `cmsh` in the `base` object under `partition` mode.

Changing the networking parameters of a cluster (apart from the IP address of the cluster) therefore requires making changes in the settings of the two preceding objects.

Adding custom DNS zones: BCM appends a BIND DNS `include` directive to the bottom of the head node name server's generated configuration file, so that administrator-defined DNS zones are preserved across regeneration. The path of both files depends on the distribution:

- on Ubuntu:
  - configuration file `/etc/bind/named.conf`
  - include file `/etc/bind/named.conf.include`
- on RHEL 9/10 and SLES 15:
  - configuration file `/etc/named.conf`
  - include file `/etc/named.conf.include`

Custom zone definitions placed in the include file are not touched by BCM. The zone file itself can be placed in any directory the administrator chooses; it is not tied to the distribution-specific `named.conf`/include file paths in the preceding. For example:

Example

```
zone "runai.superpod-nvidia.local" {
    type master;
    file "<path-to-zone-directory>/runai.superpod-nvidia.local.zone";
};
```

These and other related BIND DNS files that are managed by CMDaemon on the head node are listed in Appendix A.1.

By default, the head node name server only resolves such zones for queries from the internal network. To resolve them from other networks, the `PublicDNS` directive (Appendix C) must also be set to `true`.

The IP address of the cluster: The cluster object itself does not contain an IP address value. This is because it is the cluster network topology type that determines whether a direct interface exists from the cluster to the outside world. Thus, the IP address of the cluster in the Type 1, Type 2, and Type 3 configurations (section 3.3.9 of the *Installation Manual*) is defined by the cluster interface that faces the outside world. For Type 1, this is the interface of the head node to the external network (figure 3.2). For Type 2 and Type 3 interfaces the cluster IP address is effectively that of an upstream router, and thus not a part of BCM configuration. Thus, logically, the IP address of the cluster is not a part of the cluster object or external network object configuration.

For a Type 1 cluster, the head node IP address can be set in BCM, separately from the cluster object settings. This is then the IP address of the cluster according to the outside world.

Setting the network parameters of the cluster and the head node IP address: These values can be set using Base View or `cmsh`:

With Base View: The cluster network object and associated settings are accessed as follows:

The external network object:
The external network object is accessed via the navigation path:
`Networking > Networks > externalnet >` ⧉ `> Settings`
which reaches the window shown in figure 3.7:

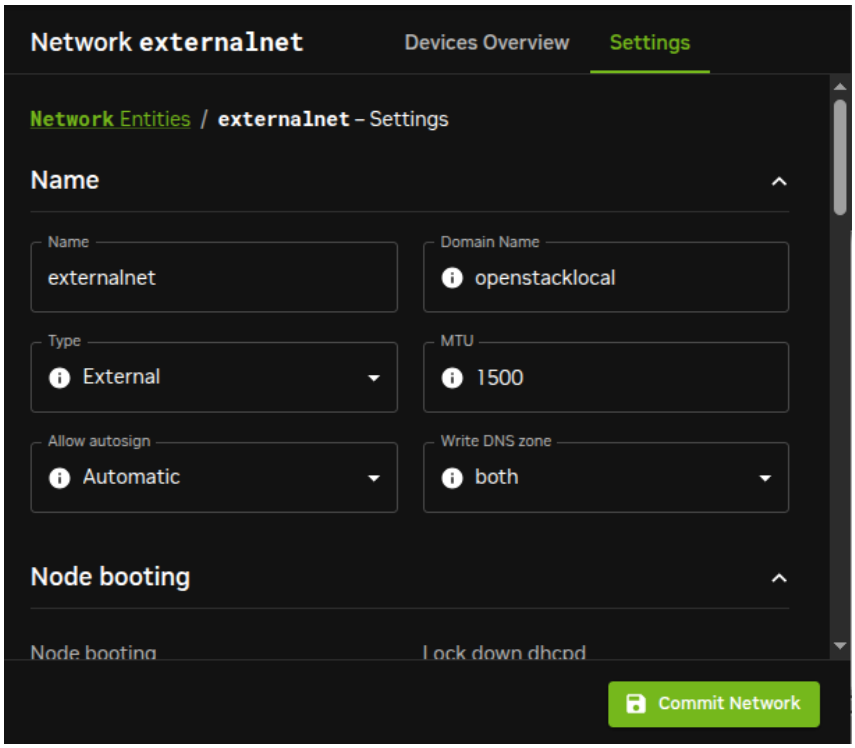


Figure 3.7: Network Settings For External Network

The cluster object:
The cluster object and associated settings are accessed using the navigation path:

`Cluster > Settings`

The head node IP address:
For a head node `basecm11`, where the IP address is used by the interface `eth0`, the static IP address can be set via the navigation path:

`Devices > All Devices > Head Node >` ⧉ `> Settings > Interfaces >` ⧉ `> eth0 >` ⧉ `> IP`

as seen in figure 3.8.

With `cmsh`: The preceding Base View configuration can also be done in `cmsh`, using the `network`, `partition` and `device` modes, as in the following example:

Example

```
[basecm11]% network use externalnet
```

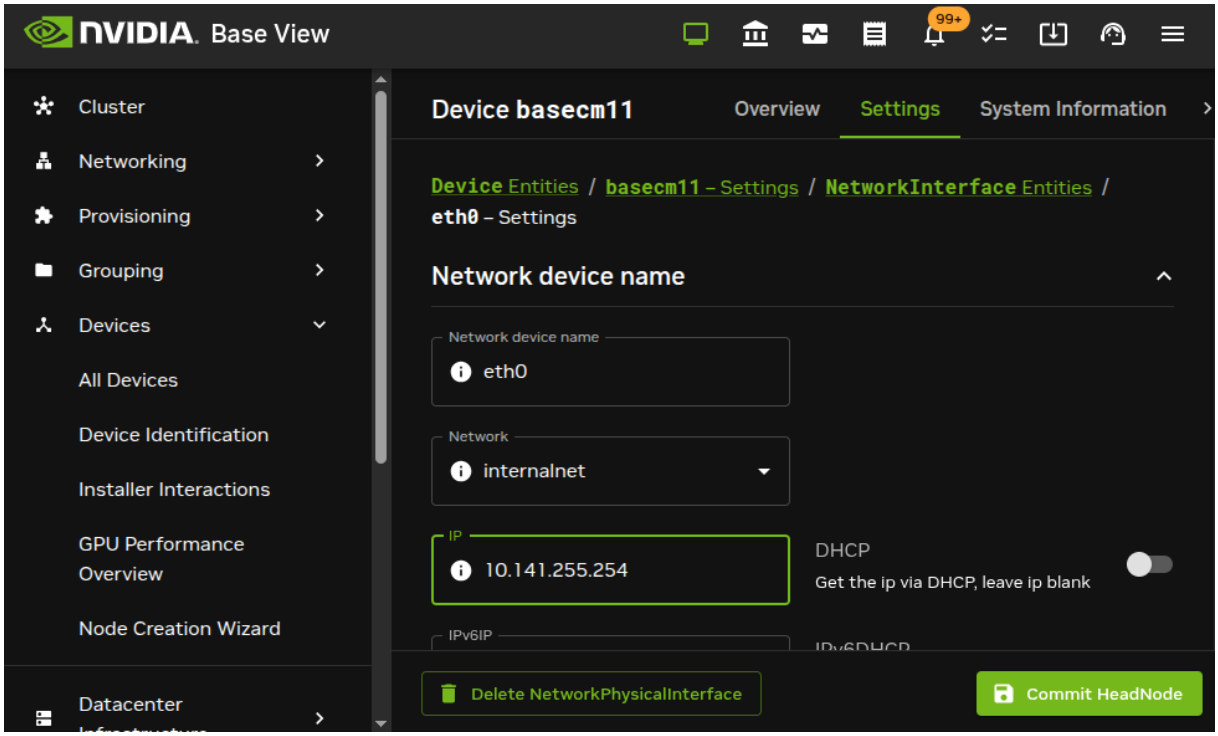


Figure 3.8: Setting The IP Address On A Head Node In Base View

```
[basecm11->network[externalnet]]% set baseaddress 192.168.1.0
[basecm11->network*[externalnet*]]% set netmaskbits 24
[basecm11->network*[externalnet*]]% set gateway 192.168.1.1
[basecm11->network*[externalnet*]]% set gatewaymetric 1
[basecm11->network*[externalnet*]]% commit
[basecm11->network[externalnet]]% partition use base
[basecm11->partition[base]]% set nameservers 192.168.1.1
[basecm11->partition*[base*]]% set searchdomains x.com y.com
[basecm11->partition*[base*]]% append timeservers ntp.x.com
[basecm11->partition*[base*]]% commit
[basecm11->partition[base]]% device use basecm11
[basecm11->device[basecm11]]% interfaces
[basecm11->device[basecm11]->interfaces]% use eth1
[basecm11->device[basecm11]->interfaces[eth1]]% set ip 192.168.1.176
[basecm11->device[basecm11]->interfaces*[eth1*]]% commit
[basecm11->device[basecm11]->interfaces[eth1]]%
```

After changing the external network configurations, a reboot of the head node is necessary to activate the changes.

Using DHCP to supply network values for the external interface: Connecting the cluster via DHCP on the external network is not generally recommended for production clusters. This is because DHCP-related issues can complicate networking troubleshooting compared with using static assignments.

For a Type 1 network, the cluster and head node can be made to use some of the DHCP-supplied external network values as follows:

- In Base View, the DHCP setting of figure 3.8 can be set to `Enabled`
- Alternatively, in `cmsh`, within interfaces mode for the head node interface, the value of the parameter DHCP can be set:

```
[basecm11->device[basecm11]->interfaces[eth0]]% set dhcp yes
```

The gateway address, the name server(s), and the external IP address of the head node are then obtained via a DHCP lease. Time server configuration for `externalnet` is not picked up from the DHCP server, having been set during installation (figure 3.8 in Chapter 3 of the *Installation Manual*). The time servers can be changed using Base View using the navigation path:

`Cluster > Settings > Time servers`

(figure 3.1), or using `cmsh` in `partition` mode as in the preceding example. The time zone can be changed similarly.

It is usually sensible to reboot after implementing these changes in order to test the changes are working as expected.

Changing Internal Network Parameters For The Cluster

When a cluster interacts with the internal network that the regular nodes and other devices are on, its connection behavior with the devices on that network is determined by settings in:

1. the internal network of the `Networks` resource (page 114)
2. the cluster network for the internal network (page 115)
3. the individual device network interface (page 115)
4. the node categories network-related items for the device (page 117), in the case of the device being a regular node.

In more detail:

1. The internal network object: has the network settings for all devices connecting to the internal network, for example, a login node, a head node via its `internalnet` interface, or a managed switch on the internal network. In Base View, the settings can be configured under the `Networking` resource, then under the `Networks` subwindow, then within the `Settings` option for the `internalnet` item (figure 3.5). In `cmsh` the settings can be configured by changing the values in the `internalnet` object within `cmsh`'s `network` mode. Parameters that can be changed include:

- the IP network parameters of the internal network (but not the internal IP address):
  - "`Base address`": the internal network address of the cluster (the "IP address of the internal network"). This is not to be confused with the IP address of the internal network interface of the head node. The default value is `10.141.0.0`.
  - "`Netmask bits`": the netmask size, or prefix-length, of the internal network, in bits. The default value is `16`.
  - `Gateway`: the default gateway route for the internal network. If unset, or 0.0.0.0 (the default), then its value is decided by the DHCP server on the head node, and nodes are assigned a default gateway route of `10.141.255.254`, which corresponds to using the head node as a default gateway route for the interface of the regular node. The effect of this parameter is overridden by any default gateway route value set by the value of `Default gateway` in the node category.
  - "`Dynamic range start`" and "`Dynamic range end`": These are the DHCP ranges for nodes. DHCP is unset by default. When set, unidentified nodes can use the dynamic IP address values assigned to the node by the node-installer. These values range by default from `10.141.128.0` to `10.141.143.255`. Using this feature is not generally recommended, in order to avoid the possibility of conflict with the static IP addresses of identified nodes.

  - `Node booting`: This allows nodes to boot from the provisioning system controlled by CMDaemon. The parameter is normally set for the management network (that is the network over which CMDaemon communicates to manage nodes) but booting can instead be carried out over a separate physical non-management network. Booting over InfiniBand is possible (section 5.1.3).
  - `Lock down dhcpd`, if set to `yes`, stops *new nodes* from booting via the network. New nodes are those nodes which are detected but the cluster cannot identify based on CMDaemon records. Details are given in Chapter 5 about booting, provisioning, and how a node is detected as new.

- the "`domain name`" of the network. This is the LAN domain, which is the domain machines on this network use as their domain. By default, set to a name based on the network interface type used on the internal network, for example `eth.cluster`. In any case, the FQDN must be unique for every node in the cluster.
- the network name, or what the internal network is called. By default, set to `internalnet`.
- The `MTU` size, or the maximum value for a TCP/IP packet before it fragments on this network. By default, set to 1500.

2. The cluster object: has other network settings that the internal network in the cluster uses. These particulars are configured in the `Settings` option of the cluster object resource in Base View (figure 3.1). The `cmsh` equivalents can be configured from the `base` object in `partition` mode. Values that can be set include:

- the "`Management network`". This is the network over which CMDaemon manages the nodes. Management means that CMDaemon on the head node communicates with CMDaemons on the other nodes. The communication includes CMDaemon calls and monitoring data. By default, the management network is set to `internalnet` for Type 1 and Type 2 networks, and `managementnet` in Type 3 networks. It is a partition-level cluster-wide setting by default. Partition-level settings can be overridden by the category level setting, and node-level settings can override category level or partition level settings.
- the "`Node name`" can be set to decide the prefix part of the node name. By default, set to `node`.
- the "`Node digits`" can be set to decide the possible size of numbers used for suffix part of the node name. By default, set to 3.
- the "`Default category`". This sets the category the nodes are in by default. By default, it is set to `default`.
- the "`Default software image`". This sets the image the nodes use by default, for new nodes that are not in a category yet. By default, it is set to `default-image`.
- the "`Name server`". This sets the name server used by the cluster. By default, it is set to the head node. The default configuration uses the internal network interface and serves only internal hosts. Internal hosts can override using this value at category level (page 117). By default external hosts cannot use this service. To allow external hosts to use the service for resolving internal hosts, the `PublicDNS` directive (Appendix C) must be set to `True`.

3. The internal IP addresses and other internal interface values: In Base View, the network for a node, such as a physical node `node001`, can be configured via a navigation path

`Devices > All Devices > Physical Node >` ⧉ `> Settings > Interfaces > BOOTIF >` ⧉

The subwindow that opens up then allows network configuration to be done for such nodes in a similar way to figure 3.8.

In `cmsh` the configuration can be done by changing properties from the `interfaces` submode that is within the `device` mode for the node.

The items that can be set include:

- the `Network device name`: By default, this is set to `BOOTIF` for a node that boots from the same interface as the one from which it is provisioned.
- the `Network`: By default, this is set to a value of `internalnet`.
- the `IP` address: By default, this is automatically assigned a static value, in the range `10.141.0.1` to `10.141.255.255`, with the first node being given the first address. Using a static IP address is recommended for reliability, since it allows nodes that lose connectivity to the head node to continue operating. The static address can be changed manually, in case there is an IP address or node ID conflict due to an earlier inappropriate manual intervention.

  Administrators who want DHCP addressing on the internal network, despite the consequences, can set it via a checkbox.
- `onnetworkpriority`: This sets the priority of DNS resolution queries for the interface on the network. The range of values that it can take is from 0 to 4294967295. Resolution takes place via the interface with the higher value.

  The default priority value for a network interface is set according to its type. These defaults are:

Type	Value
Bridge	80
Bond	70
Physical	60
VLAN	50
Alias	40
Tunnel	30
Netmap	20
BMC	10

- `Additional Hostname`: In the case of nodes this is in addition to the default node name set during provisioning. The node name set during provisioning takes a default form of `node<3 digit number>`, as explained earlier on page 115 in the section describing the cluster object settings.

For, example, a regular node that has an extra interface, `eth1`, can have its values set as follows:

Example

```
[basecm11] device interfaces node001
[basecm11->device[node001]->interfaces]% add physical eth1
[basecm11->...->interfaces*[eth1*]]% set network externalnet
[basecm11->...->interfaces*[eth1*]]% set additionalhostnames extra01
[basecm11->...->interfaces*[eth1*]]% set ip 10.141.1.1
[basecm11->...->interfaces*[eth1*]]% commit
[basecm11->...->interfaces[eth1]]% ..
[basecm11->...->interfaces]% ..
[basecm11->device[node001]]% reboot
```

4. Node category network values: are settings for the internal network that can be configured for node categories. For example, for the default category called `default` this can be carried out:

- via Base View, using the `Settings` option for the category. This is in the navigation path `Grouping > Categories >` `> Settings`
- or via the `category` mode in `cmsh`, for the `default` node category.

If there are individual node settings that have been configured in Base View or `cmsh`, then the node settings override the corresponding category settings for those particular nodes.

The category properties involved in internal networking that can be set include:

- `Default gateway`: The default gateway route for nodes in the node category. If unset, or 0.0.0.0 (the default), then the node default gateway route is decided by the internal network object `Gateway` value. If the default gateway is set as a node category value, then nodes use the node category value as their default gateway route instead.
- `Management network`: The management network is the network used by CMDaemon to manage devices. The default setting is a property of the node object. It can be set as a category property.
- `Name server`, `Time server`, `Search domain`: The default setting for these on all nodes is set by the node-installer to refer to the head node, and is not configurable at the node level using Base View or `cmsh`. The setting can however be set as a category property, either as a space-separated list of entries or as a single entry, to override the default value.

Application to generic network devices: The preceding details for the internal network parameters of the cluster, starting from page 114, are applicable to regular nodes, but they are often also applicable to generic network devices (section 2.1.1). Benefits of adding a generic device to be managed by BCM include that:

- the name given to the device during addition is automatically placed in the internal DNS zone, so that the device is accessible by name
- the device status is automatically monitored via an ICMP ping (Appendix G.2.1).
- the device can be made to work with the health check and metric framework. The scripts used in the framework will however almost certainly have to be customized to work with the device

After changing network configurations, a reboot of the device is necessary to activate the changes.

Changing The Global Network Parameters For The Cluster

The global network `globalnet` is a unique network used to set up a common name space for all nodes in a cluster in BCM. It is required in BCM because of the added cloud extension functionality, described in the *Cloudbursting Manual*. Regular nodes and regular cloud nodes are thus both named under the `globalnet` domain name, which is `cm.cluster` by default. So, for example, if default host names for regular nodes (node001, node002, ...) and regular cloud nodes (cnode001, cnode002, ...) are used, node addresses with the domain are:

- node001.cm.cluster for node001
- cnode001.cm.cluster for cnode001

The only parameter that can be sensibly changed on `globalnet` is the domain name, which is `cm.cluster` by default.

Removing `globalnet` should not be done because it will cause various networking failures, even for clusters deploying no cloud nodes.

Details on how resolution is carried out with the help of `globalnet` are given in section 6.6.1 of the *Cloudbursting Manual*.

Setting Static Routes In The `staticroutes` Submode Of `cmsh`

To route via a specific gateway, the `staticroutes` submode can be used. This can be set for regular nodes and head nodes via the `device` mode, and for node categories via the `category` mode.

On a newly-installed cluster with a type 1 network (section 3.3.9 of the *Installation Manual*), a node by default routes packets using the head node as the default gateway.

If the administrator would like to configure a regular node to use another gateway to reach a printer on another subnet, as illustrated by figure 3.9:

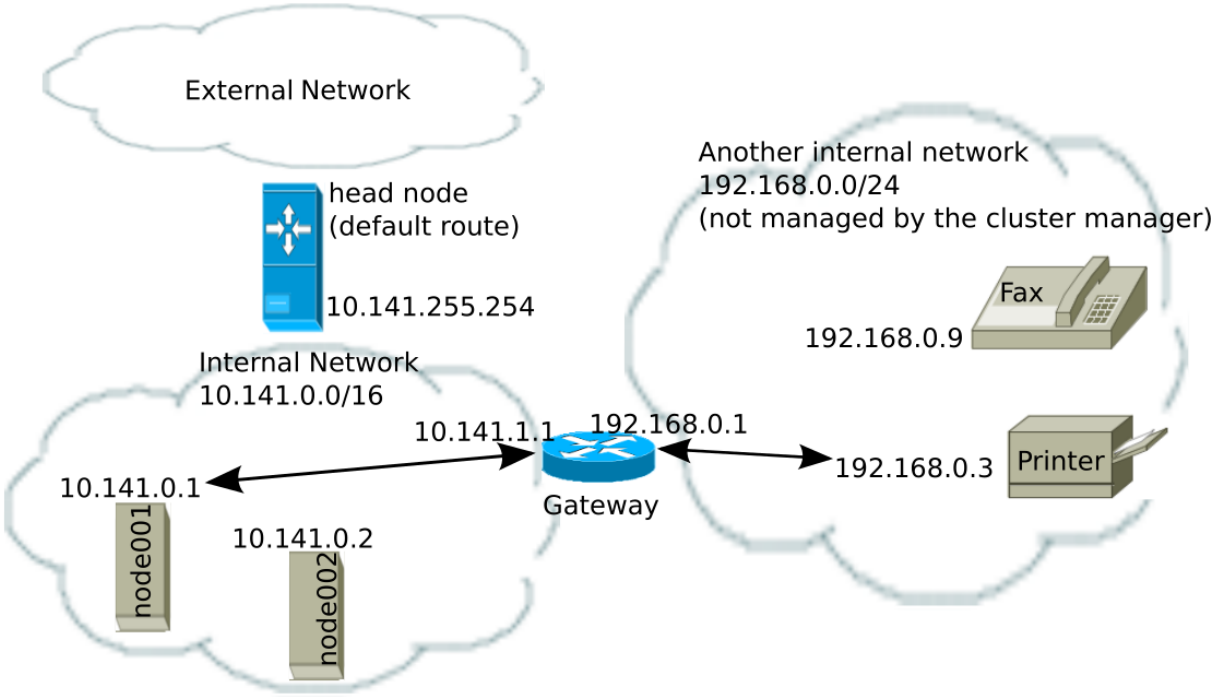


Figure 3.9: Example Of A Static Route To A Printer On Another Subnet

then an example session for setting the static route is as follows:

Example

```
[root@basecm11 ~]# cmsh
[basecm11 ]% device use node001
[basecm11->device[node001]]%
[basecm11->device[node001]]% staticroutes
[basecm11->device[node001]->staticroutes]% list
Name (key)                      Gateway          Destination
------------------------------- ---------------- --------------
[basecm11->device[node001]->staticroutes]% add printerroute
[basecm11->...*[printerroute*]]% set gateway 10.141.1.1
[basecm11->...*[printerroute*]]% set destination 192.168.0.3
[basecm11->...*[printerroute*]]% set networkdevicename bootif
[basecm11->...*[printerroute*]]% commit
[basecm11->...[printerroute]]% show
Parameter                       Value
------------------------------- ------------------------------
Destination                     192.168.0.3/32
Gateway                         10.141.1.1
Name                            printerroute
Network Device Name             bootif
```

```
Revision
[basecm11->device[node001]->staticroutes[printerroute]]% exit
[basecm11->device[node001]->staticroutes]% list
Name (key)                      Gateway          Destination
------------------------------- ---------------- --------------
printerroute                    10.141.1.1       192.168.0.3/32
[basecm11->device[node001]->staticroutes]% exit; exit
[basecm11->device]% reboot node001
```

In the preceding example, the regular node `node001`, with IP address 10.141.0.1 can connect to a host 192.168.0.3 outside the regular node subnet using a gateway with the IP addresses 10.141.1.1 and 192.168.0.1. The route is given the arbitrary name `printerroute` for administrator and CMDaemon convenience, because the host to be reached in this case is a print server. The `networkdevicename` is given the interface name `bootif`. If another device interface name is defined in CMDaemon, then that can be used instead. If there is only one interface, then `networkdevicename` need not be set.

After a reboot of the node, the route that packets from the node follow can be seen with a traceroute or similar. The `ping` utility with the `-R` option can often track up to 9 hops, and can be used to track the route:

Example

```
[root@basecm11 ~]# ssh node001 ping -c1 -R 192.168.0.3
PING 192.168.0.3 (192.168.0.3) 56(124) bytes of data.
64 bytes from 192.168.0.3: icmp_seq=1 ttl=62 time=1.41 ms
RR:     10.141.0.1
        10.141.1.1
        192.168.0.1
        192.168.0.3
        192.168.0.3
        192.168.0.1
        10.141.1.1
        10.141.0.1

--- 192.168.0.3 ping statistics ---
1 packets transmitted, 1 received, 0% packet loss, time 1ms
rtt min/avg/max/mdev = 1.416/1.416/1.416/0.000 ms
[root@basecm11->device[node001]->staticroutes]%
```

The routing is achieved by CMDaemon making sure that whenever the network interface is brought up, the OS of the regular node runs a routing command to configure the routing tables. The command executed for the given example is either:

```
route add -host 192.168.0.3 gw 10.141.1.1
```

or its modern iproute2 equivalent:

```
ip route add 192.168.0.3 via 10.141.1.1
```

If the device `bootif` is `eth0`—which is often the case—then the command is run from the network interface configuration file: `/etc/sysconfig/network-scripts/ifcfg-eth0` (or `/etc/sysconfig/network/ifcfg-eth0` in SUSE).

If the device is a head node: then rebooting it to have the new route become active is a clean way to implement the change. However doing this may not be acceptable to cluster users due to the downtime, and the associated interruption of jobs. A quicker update to the interface that should work is using operating system networking scripts. For example, if the interface with the route is `eth0`, then the interface

can be brought down and up again with `ifdown eth0 && ifup eth0`. If the various networking configurations have been applied in a way that persists over reboots, then they should be applied automatically to the interface as it starts up.

Port Forwarding With `portforward`

Why port forwarding is useful: In a typical type 1 network setup (section 3.3.9 of the *Installation Manual*), only the head nodes are accessible from the external network. So, the compute nodes cannot be accessed directly from the external network.

Accessing a compute node port may however be needed for some workflows. For example, a user on a laptop on the external network may wish to access a compute node port directly.

To allow this direct access to a compute node port, port forwarding rules can be set up from within the `device` mode of `cmsh`. Configuring port forwarding with BCM is designed to be easier than doing port forwarding directly in the operating system.

The port forwarding help text: The help text for the `portforward` command summarizes how to use it:

```
[basecm11->device]% help portforward
Name:

  portforward - Forward a port on the active head node to IP

Usage:

  portforward create <IP> <port> <active-head-node-port>
  portforward destroy [IP] [port] [active-head-node-port]
  portforward bmc [device] [https-interface-port] [console-port]
  portforward list

Examples:

  portforward create 10.148.0.1 443 8443    Create a mapping from head:8443 to 10.148.0.1:443
  portforward create node001 8081 8181      Create a mapping from head:8181 to the management IP of node001:8081
  portforward create node001:rf0 443 8443   Create a mapping from head:8443 to the redfish IP of node001:443
  portforward destroy 10.148.0.1            Destroy all port forwarding to 10.148.0.1
  portforward destroy                       Destroy all port forwarding
  portforward bmc                           Create a mapping to the BMC of the current device
  portforward bmc node001                   Create a mapping to the BMC of node001
  portforward list                          List all active mappings
```

An illustration of port forwarding working: The following example indicates how BCM port forwarding works. A laptop user on the external network may wish to connect to port 1234 on compute node `node001` by connecting to the head node port 4321. There is no port 4321 open on the head node to start with, until a `portforward` command is run:

Example

```
[basecm11->device]% connections master | grep 4321
[basecm11->device]% portforward create node001 1234 4321
[basecm11->device]% connections master | grep 4321
basecm11  TCP6     0.0.0.0                   4321     0.0.0.0         0        Listening
```

The `connections` command is described on page 124.

In the background on the active head there is a new `socat` process that is created for each port forward:

```
[basecm11->device]% !ps x | grep [s]ocat
 159559 ?        S      0:00 /usr/bin/socat tcp-listen:4321,reuseaddr,fork tcp:10.141.0.1:1234
```

The socat process forwards any packets from external network interface of the head node to the internal network interface of the compute node, as illustrated by the following ASCII diagram:

```
                external            internal
company laptop  <------>    head    <------>  node001
                <-- 4321:10.141.0.1:1234 -->
```

The forwarding can be tested by setting up a netcat listening process on port 1234 of node001:

```
[basecm11->device]% ssh node001 "nc -l 1234"
```

and then typing something to port 4321 on the head node:

```
root@basecm11:~# nc basecm11 4321
Hello, World!
^C
```

The Ctrl-C interrupts the session on the head node.

The Hello, World! text entered on the head node shows up on the netcat listener on node001. The session on the compute node ends with the Ctrl-C interrupt on the head node:

```
[basecm11->device]% ssh node001 "nc -l 1234"
Hello, World!
[basecm11->device]%
```

A successful session as just illustrated means that port forwarding is working as expected.

If the session is unsuccessful, then it may be because the port is blocked by a firewall. By default, the head node runs a firewall, Shorewall, (page 77 of the *Installation Manual*), which may need to have its rules adjusted.

Listing active port forwarding rules: All are listed with the list command:

Example

```
[basecm11->device]% portforward list
Device       IP           Port         Head port    Sessions
------------ ------------ ------------ ------------ --------
node001      10.141.0.1   8081         8181         1
node002      10.141.0.2   8081         8281         1
node003      10.141.0.3   8081         8381         1
node004      10.141.0.4   8081         8481         1
```

Sessions denotes the number of cmsh instances that started the exact same rule. This is normally 1. Each request is tracked via a cmsh session unique identifier to prevent killing a port forward rule placed by someone else.

Destroying (removing) every rule: A specific rule can be removed by specifying its compute node IP address and compute node port. All rules are removed with the destroy option used on its own:

Example

```
[basecm11->device]% portforward destroy
Destroyed: 4 port forward
```

BMC port forwarding: For BMC port forwarding 2 rules are created.

- A rule for the BMC management interface on 443
- A rule for the BMC console port (5900 or 17900 depending on the type of interface)

3.2.4 Tools For Viewing Cluster Connections And Connectivity

Viewing Node Routes With `routes` In `cmsh`

The `cmsh` `routes` command is a wrapper around the Linux `route` command, and is designed to run in parallel over nodes. It is intended to get a fast route overview for one or more nodes, and display it for easy comparison.

To get a full overview of the routes for all nodes, the command is run without any options:

Example

```
[root@basecm11 ~]# cmsh
[basecm11]% device
[basecm11->device]% routes
Hostname  Destination      Gateway          Netmask          Flags     Metric  Ref  Use  Interface
--------- ---------------- ---------------- ---------------- --------- ------- ---- ---- ---------
basecm11 0.0.0.0          192.168.255.254  0.0.0.0          UG (0003)  0       0    0    eth1
basecm11 10.141.0.0       0.0.0.0          255.255.0.0      U (0001)   0       0    0    eth0
basecm11 169.254.169.254  192.168.241.1    255.255.255.255  UGH (0007) 0       0    0    eth1
basecm11 192.168.240.0    0.0.0.0          255.255.240.0    U (0001)   0       0    0    eth1
node001  0.0.0.0          10.141.255.254   0.0.0.0          UG (0003)  0       0    0    ens3
node001  10.141.0.0       0.0.0.0          255.255.0.0      U (0001)   0       0    0    ens3
```

To select or filter the output, the grouping options of `routes`, or the text processing utility `awk` can be used. Grouping options are options to select nodes, groups, categories, and so on, and are described in the cluster management chapter, on page 72, while a handy link for `awk` one-liners is `http://tuxgraphics.org/~guido/scripts/awk-one-liner.html`.

Selection or filtering makes it very easy to detect badly-configured nodes. For example, when the expected gateway is <*expected-gateway*>, the following command may be used to display only the unexpected gateways:

Example

```
[root@basecm11 ~]# cmsh
[basecm11]% device
[basecm11->device]% routes --category default | awk '{if ($3 != "<expected-gateway>") print $0}'
...
```

Viewing IP Addresses With `ipaddresses` In `cmsh`

The `ipaddresses` command, in `device` mode, lists the configured and detected IP addresses for one or more devices. By default, link-local addresses (`169.254.0.0/16` and `fe80::/10`) are excluded from the listing, since they are rarely of interest to an administrator. The `--all` option includes them in this default listing; it has no effect on the following `--check` output, which always excludes link-local and BMC interfaces.

Example

```
[basecm11->device]% ipaddresses node001
Hostname Name IP         MAC               PCI Bus ID State
-------- ---- ---------- ----------------- ---------- -----
node001  eth0 10.141.0.1 FA:16:3E:F0:C4:EA            up
node001  eth1 10.141.1.1 FA:16:3E:B5:7D:02            up
```

The output lists one row per configured or detected address. The preceding example omits the `Netmask`, `MTU`, `Speed`, and `Duplex` columns to fit the page; the full set of columns is:

- `Hostname`: the node the interface belongs to.
- `Name`: the interface name.
- `IP`: the IP address on that interface.
- `Netmask`: the netmask for the address.
- `MAC`: the interface's MAC address.
- `PCI Bus ID`: the PCI bus address of the NIC that the IP address is assigned to, for example `0000:03:00.0`. This identifies which physical network interface corresponds to a given IP address on a node, which is useful on hosts with several NICs. It is blank for interfaces with no discoverable PCI device, such as virtual NICs, as in the preceding example. The REST API exposes the same value as the `pciBusId` field on the `IPAddress` entity.
- `MTU`: the interface's maximum transmission unit.
- `Speed`: the negotiated link speed, or `-` if not available.
- `Duplex`: the negotiated duplex mode, or `unknown` if not detected.
- `State`: whether the interface is `up` or `down`.

The `--check` option compares the IP address configured in BCM for each interface against the address actually detected on the device, and reports any mismatch:

Example

```
[basecm11->device]% ipaddresses node001 --check
Hostname Interface Configured    Detected       State
-------- --------- ------------- -------------- --------
node001  eth0      10.141.0.1    10.141.0.1     OK
node001  eth1      10.141.128.1  10.141.128.1   OK
```

If any interface is not `OK`, the command exits with a non-zero status, though the table is still printed in full:

Example

```
[basecm11->device]% ipaddresses node001 --check
Hostname Interface Configured    Detected       State
-------- --------- ------------- -------------- ---------------
node001  eth0      10.141.0.1    10.141.0.1     OK
node001  eth1      10.141.128.1  10.141.128.99  Mismatch
node001  eth2      10.141.255.2  -              Not detected
node001  eth3      -             10.2.0.5       Not configured
```

The `State` column values are:

- `OK`: the configured and detected addresses match, or the interface uses DHCP and an address was detected.
- `Mismatch`: both a configured and a detected address are present, but they differ.
- `Not detected`: an address is configured, but nothing was detected on the interface.
- `Not configured`: an address was detected, but nothing is configured in BCM for that interface.

The standard node selection options (`-n`, `-c`, `-g`, `-r`) can be used with `--check` to target a set of devices, for example `ipaddresses -r rack01 --check`. The `--check` option cannot be combined with `--import`, and `--json` is ignored when `--check` is used.

Viewing Connections In `cmsh`

The `connections` command can be run from within the `device` mode of `cmsh`. It is a parallel wrapper to view active TCP and UDP connections. That is, it runs as a simple command over multiple devices at the same time.

Running `connections` without options displays a full overview of the currently active TCP and UDP ports on all devices (output truncated):

Example

```
[basecm11->device]% connections
Node      Type     Source   Port  Destination  Port  State
--------- -------- -------- ----- ------------ ----- ----------
basecm11  TCP      0.0.0.0  22    0.0.0.0      0     Listening
basecm11  TCP      0.0.0.0  25    0.0.0.0      0     Listening
basecm11  TCP      0.0.0.0  111   0.0.0.0      0     Listening
basecm11  TCP      0.0.0.0  636   0.0.0.0      0     Listening
...
```

Filtering with `grep` can then show which nodes are listening on which ports. For example, nodes listening for DNS queries (port 53) can be found with:

Example

```
[basecm11->device]% connections | head -2; connections | grep Listen | grep " 53"
Node      Type     Source              Port     Destination     Port     State
--------- -------- ------------------ -------- --------------- -------- -----------
basecm11  TCP      10.141.255.254     53       0.0.0.0         0        Listening
basecm11  TCP      127.0.0.1          53       0.0.0.0         0        Listening
basecm11  TCP      192.168.200.162    53       0.0.0.0         0        Listening
```

and shows that only the head node is listening on that port, providing DNS.

Some third party tools require a free port on all nodes for a service to listen on. Filtering and sorting the output of the connections command allows the administrator to find all the existing used ports across all nodes:

Example

```
[basecm11->device]% connections | awk 'print $4' | sort -un
Port
22
25
53
67
68
69
...
```

Options to the `connections` command can be seen by running `help connections`. Options include node grouping options (such as `-n|--nodes`, `-c|--category`, and `-t|--type`), and filtering out TCP6 and UDP6 connections (`--no-tcp6`, `--no-udp6`).

Viewing Connections In Base View

The preceding `connections` command in `device` mode of `cmsh` has a Base View version. This is the `Network Connections` pane display, which is accessible using:

`All devices > Bulk actions > Connectivity > Network Connections`

Viewing Connectivity in `cmsh`

The `connectivity` command can be run from within the `device` mode of `cmsh`. It runs ICMP pings along each node route on a network of the cluster. By default the network is the management network.

By default the output displays if the connection is OK, the ping sequence ID (counting starts from zero), and the latency between the source and destination:

Example

```
[basecm11->device]% connectivity
Source    Destination Result       ID   Latency
--------- ----------- ------------ ---- ------------
basecm11  node001     Ok           0    0.4ms
basecm11  node002     Ok           0    0.3ms
basecm11  node003     Ok           0    0.4ms
basecm11  basecm11    Ok           0    0.1ms
```

Pings where the source and destination have identical names are carried out via identical interfaces.

Further options to the `connectivity` command can be seen by running `help connectivity`. Options include the ability to set ping timeouts and the cluster network on which to ping, as well as node grouping options.

For larger clusters, the following type of connectivity check may be a helpful diagnostic:

Example

```
[basecm11->device]% connectivity --statistics --count 100 --delay 0.01
Name        Value
----------- --------
OK          400
Total       400
Count       2
Average     0.35ms
Minimum     0.35ms
Maximum     0.35ms
Uniformity  100.0%
```

The preceding shows how uniform nodes are in ping timings. A significant spread in uniformity can indicate network problems.

Viewing Connectivity in Base View

The preceding `connectivity` command in `device` mode of `cmsh` has a Base View version. This is the `Ping All` pane display, which is accessible using:

`All devices > Bulk actions > Connectivity > Ping All`

Viewing Network Performance With `cm-iperf.py`

A handy tool for checking the network performance of the nodes on a cluster is `cm-iperf.py`.

The nodes to be checked must be specified with the `-n|--node` option:

Example

```
basecm11:~#  /cm/local/apps/cmd/scripts/cm-iperf.py  -n node001..node004
Thread  Port    Server          Client          Target          Bitrate
======= ======= =============== =============== =============== =================
0       5000    node001         node002         node001         7.89 Gbits/sec
0       5001    node001         node003         node001         7.84 Gbits/sec
0       5002    node001         node004         node001         8.22 Gbits/sec
0       5003    node002         node003         node002         7.53 Gbits/sec
0       5004    node002         node004         node002         8.60 Gbits/sec
0       5005    node003         node004         node003         7.59 Gbits/sec
```

For a Type 1 cluster (section 3.3.9 of the *Installation Manual*) with n nodes, the total number of connection pairs that can be evaluated is:

$$\binom{n}{2} = \frac{(n)(n-1)}{2}$$

For a 4-node cluster this means 6 network connection pairs are displayed, as is seen in the preceding example.

Each pair is checked for about 10s. This means that for a 1000-node cluster, the time for a default run would take about 2 months. The following options are therefore useful:

- `--count`: specify the number of pairs that are checked
- `-r|--random`: have the connection pairs chosen in a random sequence
- `-p|--parallel`: specify the number of parallel threads to runs the process

Example

```
basecm11:~#  /cm/local/apps/cmd/scripts/cm-iperf.py  -n node001..node004 --count 5 -r -p 2
```

Further options can be viewed with the `-h|--help` option.

3.3 Configuring Bridge Interfaces

Bridge interfaces can be used to divide one physical network into two separate network segments at layer 2 without creating separate IP subnets. A bridge thus connects the two networks together at layer 3 in a protocol-independent way.

The network device name given to the bridge interface is of the form `br<number>`. The following example demonstrates how to set up a bridge interface in `cmsh`, where the name `br0` is stored by the parameter `networkdevicename`.

Example

```
[basecm11->device[node001]->interfaces]% add bridge br0
[basecm11->...->interfaces*[br0*]]% set network internalnet
[basecm11->...->interfaces*[br0*]]% set ip 10.141.1.1
[basecm11->...->interfaces*[br0*]]% show
Parameter                      Value
------------------------------ ------------------
Additional Hostnames
DHCP                           no
Forward Delay                  0
IP                             10.141.1.1
Interfaces
MAC                            00:00:00:00:00:00
Network                        internalnet
Network device name            br0
Revision
SpanningTreeProtocol           no
Type                           bridge
```

A bridge interface is composed of one or more physical interfaces. The IP and network fields of the member interfaces must be empty:

Example

```
[basecm11->...->interfaces*[br0*]]% set interfaces eth1 eth2; exit
[basecm11->...->interfaces*]% clear eth1 ip; clear eth1 network
[basecm11->...->interfaces*]% clear eth2 ip; clear eth2 network
[basecm11->...->interfaces*]% use br0; commit
```

The BOOTIF interface is also a valid interface option.

Currently, the following bridge properties can be set:

- SpanningTreeProtocol: sets the spanning tree protocol to be on or off. The default is off.
- Forward Delay: sets the delay for forwarding Ethernet frames, in seconds. The default is 0.

Additional properties, if required, can be set manually using the brctl command in the OS shell.

When listing interfaces in cmsh, if an interface is a member of a bond (or bridge) interface, then the corresponding bond or bridge interface name is shown in parentheses after the member interface name:

Example

```
[headnode->device[node001]->interfaces]% list
Type         Network device name  IP              Network
------------ -------------------- --------------- ----------------
bond         bond0 [prov]         10.141.128.1    internalnet
bridge       br0                  10.141.128.2    internalnet
physical     eth0                 10.141.0.1      internalnet
physical     eth1 (bond0)         0.0.0.0
physical     eth2 (bond0)         0.0.0.0
physical     eth3 (br0)           0.0.0.0
physical     eth4 (br0)           0.0.0.0
```

It is possible to create a bridge interface with no IP address configured, that is, with an IP address of 0.0.0.0. This can be done for security reasons, or when the number of available IP addresses on the network is scarce. As long as such a bridge interface has a network assigned to it, it is properly configured on the nodes and functions as a bridge on the network.

3.4 Configuring VLAN interfaces

A VLAN (Virtual Local Area Network) is an independent logical LAN within a physical LAN network. VLAN tagging is used to segregate VLANs from each other. VLAN tagging is the practice of inserting a VLAN ID tag into a packet frame header, so that each packet can be associated with a VLAN.

The physical network then typically has sections that are VLAN-aware or VLAN-unaware. The nodes and switches of the VLAN-aware section are configured by the administrator to decide which traffic belongs to which VLAN.

A VLAN interface can be configured for an interface within BCM using cmsh or Base View.

3.4.1 Configuring A VLAN Interface Using cmsh

In the following cmsh session, a regular node interface that faces a VLAN-aware switch is made VLAN-aware, and assigned a new interface—an alias interface. It is also assigned a network, and an IP address within that network:

Example

```
[root@basecm11 ~]# cmsh
[basecm11]% device interfaces node001
[basecm11->device[node001]->interfaces]% list
Type         Network device name  IP             Network
```

```
------------ ------------------- ------------- ------------
physical     BOOTIF [prov]        10.141.0.1     internalnet
[basecm11->device[node001]->interfaces]% list -h | tail -4
Arguments:
   type
     alias, bmc, bond, bridge, netmap, physical, tunnel, vlan

[basecm11->device[node001]->interfaces]% add vlan BOOTIF.1
[basecm11->device*[node001*]->interfaces*[BOOTIF.1*]]% commit
[basecm11->device[node001]->interfaces[BOOTIF.1]]% show

Parameter                 Value
------------------------- ------------------------
Additional Hostname
DHCP                      no
IP                        0.0.0.0
MAC                       00:00:00:00:00:00
Network
Network device name       BOOTIF.1
Reorder HDR               no
Revision
Type                      vlan
[basecm11->...[BOOTIF.1]]% set network internalnet
[basecm11->...[BOOTIF.1*]]% set ip 10.141.2.1; commit
```

The `Reorder HDR` setting in a VLAN, if enabled, moves the ethernet header around to make it look exactly like a real ethernet device. This setting controls the `REORDER_HDR` setting in the file at `/etc/sysconfig/network-scripts/ifcfg-`*<interface>* on the node.

3.4.2 Configuring A VLAN Interface Using Base View

Within Base View a VLAN interface can be added for a node such as `node001` using the navigation path: `Devices > All Devices > node001` > `Settings > Interfaces` > `+ Add > vlan` (figure 3.10).

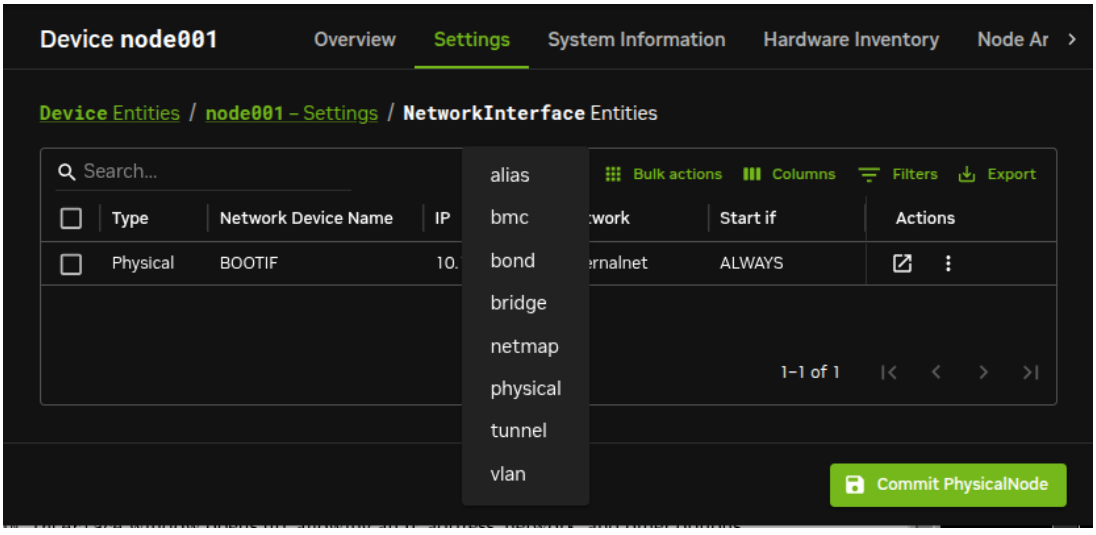


Figure 3.10: Configuring a VLAN via Base View

A `Network VLAN Interface` window opens up, allowing an IP address, network, and other options to be configured for the VLAN interface.

3.5 Configuring Bonded Interfaces

3.5.1 Adding A Bonded Interface

The Linux bonding driver allows multiple physical network interfaces that have been configured previously (for example, as on page 116) to be bonded as a single logical bond interface. The behavior of such interfaces is determined by their bonding mode. The modes provide facilities such as hot standby or load balancing.

The driver is included by default on head nodes. To configure a non-head node to use a bonded interface, a Linux kernel module called the `bonding` module must be included in the kernel modules list of the software image of the node. A bonding interface can then be created, and its properties set as follows:

Example

```
[basecm11->device[node001]->interfaces]% add bond bond0
[...->device*[node001*]->interfaces*[bond0*]]% set network internalnet
[...->device*[node001*]->interfaces*[bond0*]]% set ip 10.141.128.1
[...->device*[node001*]->interfaces*[bond0*]]% set mode 3
[...->device*[node001*]->interfaces*[bond0*]]% set interfaces eth1 eth2
```

Each bonded interface has a unique set of object properties, called bonding options, that specify how the bonding device operates. The bonding options are a string containing one or more options formatted as `option=value` pairs, with pairs separated from each other by a space character.

A bonded interface also always has a mode associated with it. By default it is set to `0`, corresponding to a balanced round-robin packet transmission.

The 7 bonding modes are:

- 0 – balance-rr
- 1 – active-backup
- 2 – balance-xor
- 3 – broadcast
- 4 – 802.3ad
- 5 – balance-tlb
- 6 – balance-alb

Technically, outside of Base View or `cmsh`, the bonding mode is just another bonding option specified as part of the options string. However in BCM the bonding mode value is set up using the dedicated `mode` property of the bonded interface, for the sake of clarity. To avoid conflict with the value of the `mode` property, trying to commit a bonding mode value as an `option=value` pair will fail validation.

3.5.2 Single Bonded Interface On A Regular Node

A single bonded interface on a node can be configured and coexist in several ways on nodes with multiple network interfaces. Possibilities and restrictions are illustrated by the following:

- The bonded interface may be made up of two member interfaces, and a third interface outside of the bond could be the boot interface. (The boot interface is the node interface used to PXE boot the node before the kernel loads (section 5.1)).
- The boot interface could itself be a member of the bonded interface. If the boot interface is a member of a bonded interface, then this is the first bonded interface when interfaces are listed as in the example on page 127.

- The bonded interface could be set up as the provisioning interface. However, the provisioning interface cannot itself be a member of a bonded interface. (The provisioning interface is the node's interface that picks up the image for the node after the initial ramdisk is running. Chapter 5 covers this in more detail).
- A bonded interface can be set up as the provisioning interface, while having a member interface which is used for PXE booting.

3.5.3 Multiple Bonded Interface On A Regular Node

A node can also have multiple bonded interfaces configured. Possibilities and restrictions are illustrated by the following:

- Any one of the configured bonded interfaces can be configured as the provisioning interface. However, as already mentioned in the case of single bond interfaces (section 3.5.2), a particular member of a bonded interface cannot be made the provisioning interface.
- When a bonded interface is set as the provisioning interface, then during the node-installer phase of boot, the node-installer brings up the necessary bonded interface along with all its member interfaces so that node provisioning is done over the bonded interface.

3.5.4 Bonded Interfaces On Head Nodes And HA Head Nodes

It is also possible to configure bonded interfaces on head nodes.

For a single head node setup, this is analogous to setting up bonding on regular nodes.

For a high availability (HA) setup (Chapter 15), bonding is possible for `internalnet` as well as for `externalnet`, but it needs the following extra changes:

- For the bonded interface on `internalnet`, the shared internal IP alias interface name (the value of `networkdevicename`, for example, `eth0:0` in figure 15.1) for that IP address should be renamed to the bonded alias interface name on `internalnet` (for example, `bond0:0`).
- For the bonded interface on `externalnet`, the shared external IP alias interface name (the value of `networkdevicename`, for example, `eth1:0` in figure 15.1) for that IP address should be renamed to the bonded alias interface name on `externalnet` (for example, `bond1:0`).
- Additionally, when using a bonded interface name for the internal network, the value of the provisioning network interface name (the value of `provisioninginterface`, for example, `eth0`) for the head nodes, must be changed to the name of the bonded interface (for example, `bond0 [prov]`) on the internal network. The `provisioninginterface` value setting is described further on page 284.

Example

```
[headnode1->device[headnode1]->interfaces]% list
Type         Network device name  IP               Network
------------ -------------------- ---------------- ------------
alias        bond0:0              10.141.255.252   internalnet
alias        bond1:0              10.150.57.1      externalnet
bond         bond0 [prov]         10.141.255.254   internalnet
bond         bond1                10.150.57.3      externalnet
physical     eth0 (bond0)         0.0.0.0
physical     eth1 (bond0)         0.0.0.0
physical     eth2 (bond1)         0.0.0.0
physical     eth3 (bond1)         0.0.0.0
```

3.5.5 Tagged VLAN On Top Of a Bonded Interface

It is possible to set up a tagged VLAN interface on top of a bonded interface. There is no requirement for the bonded interface to have an IP address configured in this case. The IP address can be set to `0.0.0.0`, however a network must be set.

Example

```
[headnode1->device[node001]->interfaces]% list
Type         Network device name  IP              Network
------------ -------------------- --------------- ------------
bond         bond0                0.0.0.0         internalnet
physical     eth0 [prov]          10.141.0.1      internalnet
physical     eth1 (bond0)         0.0.0.0
physical     eth2 (bond0)         0.0.0.0
vlan         bond0.3              10.150.1.1      othernet
```

3.5.6 Association Of MAC Address With A Bonded Interface

BCM has MAC settings for interfaces as well as for nodes.

When a node is provisioned via PXE booting (Chapter 5), the MAC on the NIC that is being booted from is matched with the MAC property of the node. This is how the node is identified.

If a bonded interface is used, then any interface may be used for PXE booting. Each interface has its own MAC address, which means that the interface MAC address may not match the assigned node MAC address. In BCM 11 and later versions, the node-installer recognizes any MAC address in a bonded interface.

3.5.7 Further Notes On Bonding

If using bonding to provide failover services, then the kernel module option setting for media independent interface monitoring, `miimon`, which is set to be off by default (set to 0), should be given a non-zero value.

The `miimon` setting is the time period in milliseconds between checks of the interface carrier state. A common value is `miimon=100`.

Its value is set in the bonding device configuration file.

- For RHEL and derivatives, and for SLES distributions, the file is in a network script along a file path such as `/etc/sysconfig/network-scripts/ifcfg-bond0`.

  The line within the stanza for the `ifcfg-bond0` file might look like:

  Example

  ```
  BONDING_MODULE_OPTS="mode=active-backup miimon=100"
  ```

  and can be set manually.

  - For RHEL and derivatives, it can alternatively be configured with front-end tools such as the `nmcli` tool, `nmtui`, or others.

    Example

    ```
    # nmcli connection add type bond con-name bond0 ifname bond0 bond.options \
      "mode=active-backup,miimon=100"
    ```

  - For SLES, YaST can be used as a front-end tool (`YaST` > `System` > `Network Settings`).

- For Ubuntu, the interface definitions are in a file, either `/etc/network/interfaces`, or a file under `/etc/network/interfaces.d/`.

  The line to set the value of `miimon` follows a form such as:

  ```
  bond-miimon 100
  ```

  instead of

  ```
  miimon=100
  ```

  and can be set manually.

  Alternatively, Canonical's `netplan` (`https://netplan.io`) utility can be used to set the network configuration files. The `netplan` YAML configuration key to set `bond-miimon` is `mii-monitor-interval`.

When listing interfaces in `cmsh`, if an interface is a member of a bond or bridge interface, then the corresponding bonded or bridge interface name is shown in parentheses after the member interface name. Section 3.3, on configuring bridge interfaces, shows an example of such a listing from within `cmsh` on page 127.

More on bonded interfaces (including a detailed description of bonding options and modes) can be found at `http://www.kernel.org/doc/Documentation/networking/bonding.txt`.

3.6 Configuring InfiniBand Interfaces

On clusters with an InfiniBand interconnect, the InfiniBand Host Channel Adapter (HCA) in each node must be configured before it can be used.

This section describes how to set up the InfiniBand service on the nodes for regular use. Setting up InfiniBand for booting and provisioning purposes is described in Chapter 5, while setting up InfiniBand for NFS is described in section 3.14.4.

3.6.1 Installing Software Packages

On a standard NVIDIA Base Command Manager cluster, the OFED (OpenFabrics Enterprise Distribution) packages that are part of the Linux base distribution are used. These packages provide RDMA implementations allowing high bandwidth/low latency interconnects on OFED hardware. The implementations can be used by InfiniBand hardware.

By default, all relevant OFED packages are installed on the head node and software images. It is possible to replace the distribution OFED with a non-distribution OFED. The replacement can be for the entire cluster, or only for certain software images. Administrators may choose to switch to a different OFED version if the HCAs used are not supported by the distribution OFED version, or to increase performance by using an OFED version that has been optimized for a particular HCA. Installing a DOCA Mellanox OFED stack onto BCM is covered in Chapter 10 of the *Installation Manual*.

If the InfiniBand network is enabled during cluster installation, then the `infiniband.conf` and `rdma.conf` modules in the subdirectory `etc/rdma/modules/` are automatically configured for the detected hardware.

The relevant InfiniBand HCA kernel modules are then automatically loaded during the init stage by systemd. Verifying that InfiniBand is active can be done after the cluster is up and running by running `ibstat`:

```
auser@head:~$ ibstat
CA 'mlx5_0'
        CA type: MT4123
        Number of ports: 1
        Firmware version: 20.31.2006
        Hardware version: 0
```

```
        Node GUID: 0xb8599f0300e4222a
        System image GUID: 0xb
        Port 1:
                State: Active
                Physical state: LinkUp
...
```

3.6.2 Mellanox Adapter Settings With `mlxconfig`

The device mode `mlxconfig` command displays and sets Mellanox adapter configuration on nodes. It is useful for checking the runtime state that the NVIDIA `mlxconfig` utility reports, and for applying selected supported settings from BCM.

Example

```
[basecm11->device]% mlxconfig node001 --adapters
Hostname  Interface  PCI           HCA     Card Name              Port
--------  ---------  ------------  ------  ---------------------  ----
node001   ib0        0000:03:00.0  mlx5_0  MCX653105A-HDA_Ax_Bx   1
```

Without `--adapters`, the command expands each matching adapter into the known mlxconfig fields. The output shows the BCM field name, the current value, the pending value, and whether a reboot is required for pending changes to take effect. The output can be limited to an interface, PCI address, port, or detail level:

Example

```
[basecm11->device]% mlxconfig node001 --interface ib0
[basecm11->device]% mlxconfig node001 --pci-address 0000:03:00.0 --port 1
[basecm11->device]% mlxconfig node001 --detail-level 1
```

To set a value immediately on a node, use `--set`. A per-port field can be selected either from the interface context or by using a port suffix in the field name:

Example

```
[basecm11->device]% mlxconfig node001 --interface ib0 --set link_type=ETH
[basecm11->device]% mlxconfig node001 --pci-address 0000:03:00.0 --port 1 --set link_type=ETH
[basecm11->device]% mlxconfig node001 --pci-address 0000:03:00.0 --set link_type_p1=ETH
```

Some mlxconfig changes are pending until the node is rebooted. When this is the case, the set result indicates that a reboot is required.

Persistent mlxconfig Policy With Extra Values

Persistent Mellanox adapter settings can be declared through `extra_values` entries whose keys start with `MLX_`. The desired settings can be defined on a category, on a node, or on a network interface. When the same field is defined at more than one level, the precedence is:

```
category.extra_values  <  node.extra_values  <  interface.extra_values
```

Category and node settings apply to all discovered Mellanox adapters on the node. Interface settings apply to the selected interface. For VLAN and bond interfaces, BCM resolves the setting to the underlying physical interface or interfaces before applying it.

The key name after `MLX_` can use a BCM field name, such as `link_type_p1`, or a native mlxconfig field name, such as `KEEP_ETH_LINK_UP_P1`. Values may be strings, booleans, or numbers. Boolean values are applied to mlxconfig as `1` or `0`.

Example

```
[basecm11->category[default]]% set -e MLX_link_type_p1 ETH
[basecm11->category*[default*]]% commit
[basecm11->device[node001]]% set -e MLX_num_vfs 16
[basecm11->device[node001]->interfaces[ib0]]% set -e MLX_KEEP_ETH_LINK_UP_P1 true
[basecm11->device*[node001*]->interfaces*[ib0*]]% commit
```

The settings are reconciled by the node after it connects to the head node and again during the periodic node update. The apply step is idempotent: BCM does not set a field again if the current or pending mlxconfig value already matches the desired value. If the local mlxconfig binary is missing, no Mellanox adapter is discovered, or mlxconfig rejects a value, BCM raises a warning event.

The `--desired` option shows the declared policy from the cluster configuration. It does not query the node or run mlxconfig, so it can be used to inspect the intended policy even before the node has applied it.

Example

```
[basecm11->device]% mlxconfig node001 --desired
Hostname  Source          Interface  Field                Value
--------  -------------   ---------  -------------------  -----
node001   category:gpu    *          link_type_p1         ETH
node001   node            *          num_vfs              16
node001   interface:ib0   ib0        KEEP_ETH_LINK_UP_P1  true
```

3.6.3 Subnet Managers

Every InfiniBand subnet requires at least one subnet manager to be running. The subnet manager takes care of routing, addressing and initialization on the InfiniBand fabric. Some InfiniBand switches include subnet managers. However, on large InfiniBand networks or in the absence of a switch-hosted subnet manager, a subnet manager needs to be started on at least one node inside of the cluster. When multiple subnet managers are started on the same InfiniBand subnet, one instance will become the active subnet manager whereas the other instances will remain in passive mode. It is recommended to run 2 subnet managers on all InfiniBand subnets to provide redundancy in case of failure.

On a Linux machine that is not running BCM, an administrator sets a subnet manager service (usually `opensm`, but `opensmd` in SLES) to start at boot-time with a command such as:

```
systemctl enable opensm.service
```

However, for clusters managed by BCM, a subnet manager is best set up using CMDaemon. There are two ways of setting CMDaemon to start up the subnet manager on a node at boot time:

1. by assigning a role.

   In `cmsh` this can be done with:

   ```
   [root@basecm11 ~]# cmsh -c "device roles <node>; assign subnetmanager; commit"
   ```

   where `<node>` is the name of a node on which it will run, for example: `basecm11`, `node001`, `node002`...

   In Base View, the subnet manager role is assigned by selecting a head node or regular node from the `Devices` resource, and assigning it the "`Subnet manager role`". The navigation path for this, for a node `node002` for example, is:

   `Devices > All Devices > node002` > `Settings` > `Roles` > `+ Add` > `[SubnetManagerRole]`

2. by setting the service up. Services are covered more generally in section 3.15.

   In `cmsh` this is done with:

Example

```
[root@basecm11 ~]# cmsh
[basecm11]% device services node001
[basecm11->device[node001]->services]% add opensm
[basecm11->device[node001]->services*[opensm*]]% set autostart yes
[basecm11->device[node001]->services*[opensm*]]% set monitored yes
[basecm11->device[node001]->services*[opensm*]]% commit
[basecm11->device[node001]->services[opensm]]%
```

In Base View the subnet manager service is configured by selecting a head node or regular node from the resources tree, and adding the service to it. The navigation path for this, for a node node002 for example, is:

Devices > All Devices > node002 > Settings > Services > + Add

When the head node in a cluster is equipped with an InfiniBand HCA, it is a good candidate to run as a subnet manager for smaller clusters.

On large clusters a dedicated node is recommended to run the subnet manager.

3.6.4 InfiniBand Network Settings

Although not strictly necessary, it is recommended that InfiniBand interfaces are assigned an IP address (i.e. IP over IB). First, a network object in the cluster management infrastructure should be created. The procedure for adding a network is described in section 3.2.2. The following settings are recommended as defaults:

Property	Value
Name	ibnet
Domain name	ib.cluster
Type	internal
Base address	10.149.0.0
Netmask bits	16
MTU	up to 4k in datagram mode
	up to 64k in connected mode

By default, an InfiniBand interface is set to datagram mode, because it scales better than connected mode. It can be configured to run in connected mode by setting the connectedmode property:

Example

```
[basecm11->device[node001]->interfaces[ib0]]% set connectedmode yes
```

For nodes that are PXE booting or are getting provisioned over InfiniBand, the mode setting in the node-installer script has to be changed accordingly.

Example

```
[root@basecm11 ~]# echo datagram > /cm/node-installer/scripts/ipoib_mode
```

Once the network has been created all nodes must be assigned an InfiniBand interface on this network. The easiest method of doing this is to create the interface for one node device and then to clone that device several times.

For large clusters, a labor-saving way to do this is using the addinterface command (section 3.7.1) as follows:

```
[root@basecm11 ~]# echo "device
addinterface -n node001..node150 physical ib0 ibnet 10.149.0.1
commit" | cmsh -x
```

When the head node is also equipped with an InfiniBand HCA, it is important that a corresponding interface is added and configured in the cluster management infrastructure.

Example

Assigning an IP address on the InfiniBand network to the head node:

```
[basecm11->device[basecm11]->interfaces]% add physical ib0
[basecm11->device[basecm11]->interfaces*[ib0*]]% set network ibnet
[basecm11->device[basecm11]->interfaces*[ib0*]]% set ip 10.149.255.254
[basecm11->device[basecm11]->interfaces*[ib0*]]% commit
```

As with any change to the network setup, the head node needs to be restarted to make the above change active.

3.6.5 Verifying Connectivity

After all nodes have been restarted, the easiest way to verify connectivity is to use the `ping` utility

Example

Pinging node015 while logged in to node014 through the InfiniBand interconnect:

```
[root@node014 ~]# ping node015.ib.cluster
PING node015.ib.cluster (10.149.0.15) 56(84) bytes of data.
64 bytes from node015.ib.cluster (10.149.0.15): icmp_seq=1 ttl=64
time=0.086 ms
...
```

If the ping utility reports that ping replies are being received, the InfiniBand is operational. The `ping` utility is not intended to benchmark high speed interconnects. For this reason it is usually a good idea to perform more elaborate testing to verify that bandwidth and latency are within the expected range.

The quickest way to stress-test the InfiniBand interconnect is to use the Intel MPI Benchmark (IMB), which is installed by default in `/cm/shared/apps/imb/current`. The `setup.sh` script in this directory can be used to create a template in a user's home directory to start a run.

Example

Running the Intel MPI Benchmark using `openmpi` to evaluate performance of the InfiniBand interconnect between `node001` and `node002`:

```
[root@basecm11 ~]# su - cmsupport
[cmsupport@basecm11 ~]$ cd /cm/shared/apps/imb/current/
[cmsupport@basecm11 current]$ ./setup.sh
[cmsupport@basecm11 current]$ cd ~/BenchMarks/imb/2017
[cmsupport@basecm11 2017]$ module load openmpi/gcc
[cmsupport@basecm11 2017]$ module initadd openmpi/gcc
[cmsupport@basecm11 2017]$ make -f make_mpi2
[cmsupport@basecm11 2017]$ mpirun -np 2 -machinefile ../nodes IMB-MPI1 PingPong
#---------------------------------------------------
# Benchmarking PingPong
# #processes = 2
#---------------------------------------------------
       #bytes #repetitions      t[usec]   Mbytes/sec
```

```
     0         1000         0.78         0.00
     1         1000         1.08         0.88
     2         1000         1.07         1.78
     4         1000         1.08         3.53
     8         1000         1.08         7.06
    16         1000         1.16        13.16
    32         1000         1.17        26.15
    64         1000         1.17        52.12
   128         1000         1.20       101.39
   256         1000         1.37       177.62
   512         1000         1.69       288.67
  1024         1000         2.30       425.34
  2048         1000         3.46       564.73
  4096         1000         7.37       530.30
  8192         1000        11.21       697.20
 16384         1000        21.63       722.24
 32768         1000        42.19       740.72
 65536          640        70.09       891.69
131072          320       125.46       996.35
262144          160       238.04      1050.25
524288           80       500.76       998.48
1048576          40      1065.28       938.72
2097152          20      2033.13       983.71
4194304          10      3887.00      1029.07


# All processes entering MPI_Finalize
```

To run on nodes other than `node001` and `node002`, the `../nodes` file must be modified to contain different hostnames. To perform other benchmarks, the `PingPong` argument should be omitted.

3.7 Configuring BMC (IPMI/iLO/DRAC/CIMC/Redfish) Interfaces

BCM can initialize and configure the baseboard management controller (BMC) that may be present on devices. This ability can be set during the installation on the head node (figure 3.15 of the *Installation Manual*), or it can be set after installation as described in this section. The IPMI, iLO, DRAC, CIMC, or Redfish interface that is exposed by a BMC is treated in the cluster management infrastructure as a special type of network interface belonging to a device. In the most common setup a dedicated network (i.e. IP subnet) is created for BMC communication. The `10.148.0.0/16` network is used by default for BMC interfaces by BCM.

3.7.1 BMC Network Settings

The first step in setting up a BMC is to add the BMC network as a network object in the cluster management infrastructure. The procedure for adding a network is described in section 3.2.2. The following settings are recommended as defaults:

Property	Value
Name	bmcnet, ilonet, ipminet, dracnet, cimcnet, or rfnet
Domain name	bmc.cluster, ilo.cluster, ipmi.cluster, drac.cluster, cimc.cluster, or rf.cluster
Type	Internal
Base address	10.148.0.0
Netmask bits	16
Broadcast address	10.148.255.255

Once the network has been created, all nodes must be assigned a BMC interface, of type bmc, on this network. The easiest method of doing this is to create the interface for one node device and then to clone that device several times.

For larger clusters this can be laborious, and a simple bash loop can be used to do the job instead:

```
[basecm11 ~]# for ((i=1; i<=150; i++)) do
echo "
device interfaces node$(printf '%03d' $i)
add bmc ipmi0
set network bmcnet
set ip 10.148.0.$i
commit"; done | cmsh -x     # -x usefully echoes what is piped into cmsh
```

The preceding loop can conveniently be replaced with the addinterface command, run from within the device mode of cmsh:

```
[basecm11 ~]# echo "
device
addinterface -n node001..node150 bmc ipmi0 bmcnet 10.148.0.1
commit" | cmsh -x
```

The help text in cmsh gives more details on how to use addinterface.

Most administrators are likely to simply run it as an interactive session in cmsh, running the help addinterface command for reference, and then supplying the options for the nodes and interface settings in device mode. For example, as in the following session:

```
[root@basecm11 ~]# cmsh
[basecm11]% device
[basecm11->device]% help addinterface
Name:
          addinterface - Add a network interface to one or more nodes

Usage:
          addinterface [OPTIONS] <type> <devicename> <network> <firstip>

Options:
...help text omitted...
Examples:
          addinterface -n node001..node010 physical ib0 ibnet 10.149.0.1

[basecm11->device]% addinterface -n node001..node150 bmc ipmi0 bmcnet 10.148.0.1
[basecm11->device*]% commit
```

In order to be able to communicate with the BMC interfaces, the head node also needs an interface on the BMC network. Depending on how the BMC interfaces are physically connected to the head node, the head node has to be assigned an IP address on the BMC network one way or another. There are two possibilities for how the BMC interface is physically connected:

- When the BMC interface is connected to the primary internal network, the head node should be assigned an alias interface configured with an IP address on the BMC network.
- When the BMC interface is connected to a dedicated physical network, the head node must also be physically connected to this network. A physical interface must be added and configured with an IP address on the BMC network.

Example

Assigning an IP address on the BMC network to the head node using an alias interface:

```
[basecm11->device[basecm11]->interfaces]% add alias eth0:0
[basecm11->device[basecm11]->interfaces*[eth0:0*]]% set network bmcnet
[basecm11->device[basecm11]->interfaces*[eth0:0*]]% set ip 10.148.255.254
[basecm11->device[basecm11]->interfaces*[eth0:0*]]% commit
[basecm11->device[basecm11]->interfaces[eth0:0]]%
Mon Dec 6 05:45:05 basecm11: Reboot required: Interfaces have been modified
[basecm11->device[basecm11]->interfaces[eth0:0]]% ..;..
[basecm11->device[basecm11]]% reboot
```

As with any change to the network setup, the head node needs to be restarted to make the above change active.

BMC connectivity from the head node to the IP addresses of the configured interfaces on the regular nodes can be tested with Bash one-liner such as:

Example

```
[root@basecm11 ~]# for i in $(cmsh -c "device; foreach -t physicalnode (interfaces; \
use ilo0; get ip)"); do ping -c1 $i; done | grep -B1 packet
--- 10.148.0.1 ping statistics ---
1 packets transmitted, 0 received, 100% packet loss, time 0ms
--- 10.148.0.2 ping statistics ---
1 packets transmitted, 0 received, 100% packet loss, time 0ms
--- 10.148.0.3 ping statistics ---
1 packets transmitted, 0 received, 100% packet loss, time 0ms
...
```

In the preceding example the packet loss demonstrates there is a connection problem between the head node and the BMC subnet.

3.7.2 BMC Authentication

The node-installer described in Chapter 5 is responsible for the initialization and configuration of the BMC interface of a device. In addition to a number of network-related settings, the node-installer also configures BMC authentication credentials. By default BMC interfaces are configured with username `bright` and a random password that is generated during the installation of the head node. The password is stored by CMDaemon. It can be managed from `cmsh` from within the `base` object of `partition` mode, in the `bmcsettings` submode. This means that by default, each BMC in the cluster has that username and password set during node boot.

For example, the current values of the BMC username and password for the entire cluster can be obtained and changed as follows:

Example

```
[basecm11]% partition use base
[basecm11->partition[base]]% bmcsettings
[basecm11->partition[base]->bmcsettings]% get username
```

```
bright
[basecm11->partition[base]->bmcsettings]% get password
Za4ohni1ohMa2zew
[basecm11->partition[base]->bmcsettings]% set username bmcadmin
[basecm11->partition*[base*]->bmcsettings*]% set password
enter new password: ******
retype new password: ******
[basecm11->partition*[base*]->bmcsettings*]% commit
```

In Base View, selecting the cluster item in the resources pane, and then using the `Settings` option, allows the BMC settings to be edited.

The BMC authentication credentials, and also some other BMC properties can be set cluster-wide, category, or per node. As usual, category settings override cluster-wide settings, and node settings override category settings. The relevant properties are:

Property	Description
`BMC User ID`	User type. Normally set to `4` for administrator access.
`BMC User Name`	User name used when sending a BMC command
`BMC Password`	Password for specified user name when sending a BMC command
`BMC Power reset delay`	Delay, in seconds, before powering up (default value: `0`)
`BMC extra arguments`	Extra arguments passed to BMC commands
`BMC privilege`	Possible options are • `administrator` • `callback` • `OEMproprietary` • `operator` • `user`

Viewing The Effective BMC Settings (`-s|--show`)

Because BMC settings can be set cluster-wide, per category, and per node, it is not always obvious which values a particular node actually uses. The `bmcsettings` command therefore accepts a `-s|--show` option that prints the effective settings without entering the `bmcsettings` submode.

When `--show` is used, BCM resolves the settings along the inheritance chain. For a node this is the node first, then its category, then the `base` partition. For a category it is the category first, then the `base` partition. The first level that has the settings defined is displayed, and a comment line indicates where the displayed values come from. If no level has the settings defined, then `< not set >` is shown instead.

```
[basecm11->device[node001]]% bmcsettings --show
# category:default
Parameter                      Value
------------------------------ ------------------------------------------------
User name                      bmcadmin
Password                       ********
User ID                        4
Privilege                      administrator
...
```

The same `-s|--show` option is available for the other settings submodes that hold a single settings object and follow the same inheritance, such as `accesssettings`, `dpusettings`, `ztpsettings`, `snmpsettings`, `selinuxsettings`, and `proxysettings`. Submodes that hold a list of objects, such as `roles` or `interfaces`, do not support it. Availability can also depend on the device type: `snmpsettings`, for example, applies to switches, rack sensors, and power distribution units, so running `snmpsettings --show` on a regular node displays the command usage instead of settings.

BMC configuration on a head node is done directly.

For regular nodes BCM stores the BMC configuration, and uses it:

- to configure the BMC interface from the node-installer
- to authenticate to the BMC interface after it has come up

BMC management operations, such as power cycling nodes and collecting hardware metrics, can then be performed after the node has been provisioned again.

If BMC authentication fails, then an explanation for why can often be found in the node-installer log at `/var/log/node-installer`.

The `credentialsoverview` command (section 3.7.3) within `device` mode is the recommended utility for seeing which BMC username and password each device ends up using, once the device → category → partition inheritance is resolved.

3.7.3 Auditing Cluster Credentials With The `credentialsoverview` Command

The `credentialsoverview` command within `device` mode lists the effective Access and BMC credentials per device.

It walks the device → category → partition inheritance chain, so that the cluster administrator can see which of the levels actually supplies the setting for the Access and BMC credentials.

Access settings are described in section 3.9.1 for Cumulus switches, but are also used cluster-wide. The BMC settings are described in section 3.7.2.

If used without arguments, then the command lists the output for every device on the cluster. The usual device-selection flags, such as a hostname list or range, `-c|--category`, `-t|--type`, `-g|--group`, `-r|--rack`, and so on, can be used to narrow the list to a specific subset.

The password columns never show plaintext. Instead they show a short SHA-256 fingerprint, salted with the running `cmsh` session's PID and start time, so that:

- within a single `cmsh` session, identical underlying passwords render as identical hex blobs, which is the whole point: two devices with diverging passwords stand out immediately
- across `cmsh` sessions, the same password produces a different fingerprint, so that a hex blob copied from someone else's scrollback cannot be used to recognise a password on a different login.
- Empty password fields render as:
  - `<not set>` for settings objects that exist and have no credential values supplied by a level
  - `-` for settings objects that do not yet exist, or for objects that are redacted by CMDaemon, in cases when the `cmsh` user lacks the `ACCESS_SETTINGS_TOKEN` or `BMC_USERNAME_PASSWORD_TOKEN` profile token

Example

```
[basecm11->device]% credentialsoverview node001..node003 node001-dpu
Device       Access user      Access pass   BMC user                    BMC pass
node001      -                -             root (category: default)    4b9c2d1e8a3f
node002      -                -             root (category: default)    4b9c2d1e8a3f
node003      -                -             root (category: default)    9e7f3a2c1b0d
node001-dpu  ubuntu (device)  c0ffee123456  root (category: dpu)        7d8a9b4c5e6f
```

In the preceding example, node003 is the odd one out because its BMC password fingerprint differs from the other members of its category.

Example

```
[basecm11->device]% credentialsoverview node003..node004
Device       Access user      Access pass   BMC user                      BMC pass
node003      -                -             root (category: default)      9e7f3a2c1b0d
node004      -                -             <not set> (partition: base)   <not set>
```

In the preceding example, node004 has not had its BMC user or password set, even though the BMC object exists.

The --inverse flag flips the table so that each distinct (access_user, access_pass, bmc_user, bmc_pass) tuple is shown once, with the matching hostnames listed beside it. Rows are sorted ascending by count, so outliers land at the top of the output:

Example

```
[basecm11->device]% credentialsoverview --inverse -t node
Count  Access user      Access pass   BMC user  BMC pass      Devices
1      -                -             root      9e7f3a2c1b0d  node003
1      ubuntu (device)  c0ffee123456  root      7d8a9b4c5e6f  node001-dpu
2      -                -             root      4b9c2d1e8a3f  node001, node002
```

Standard table-formatting flags also apply: -d|--delimiter sets a custom column separator ({} or {<digits>} is used for JSON output, just like for other cmsh list commands), and -v|--verbose enables multi-line wrapping, which is useful when a single tuple in --inverse mode covers a long list of hostnames.

3.7.4 Interfaces Settings

Interface Name

It is recommended that the network device name of a BMC interface start with ipmi, ilo, drac, cimc, or rf, according to whether the BMC is running with IPMI, iLO, DRAC, CIMC, or Redfish. Numbers are appended to the base name, resulting in, for example: ipmi0.

Obtaining The IP address

BMC interfaces can have their IP addresses configured statically, or obtained from a DHCP server.

Only a node with a static BMC IP address has BMC power management done by BCM. If the node has a DHCP-assigned BMC IP address, then it requires custom BMC power management (section 4.1.4) due to its dynamic nature.

Dell OpenManage And racadm Installation

The Dell OpenManage utilities are provided with BCM only for RHEL8-based distributions at the time of writing of this section (September 2023).

If Dell was chosen as the hardware vendor when the BCM ISO was created for installation, and chosen as the hardware manufacturer when the head node was configured in the BCM installer (section 3.3.10 of the *Installation Manual*), then the Dell OpenManage utilities are located under /opt/dell on the head node.

If Dell was chosen as the hardware manufacturer when nodes are configured in the BCM installer (section 3.3.11 of the *Installation Manual*), then the default software image that is used by the node has the Dell OpenManage utilities, located on the head node at /cm/images/default-image/opt/dell.

The Dell OpenManage utilities contain the racadm binary to carry out remote access control administration. The racadm tool can be used to issue power commands (Chapter 4). BCM runs commands similar to the following to issue the power commands:

```
/opt/dell/srvadmin/sbin/racadm -r <DRAC interface IP address> -u <bmcusername> -p <bmcpassword>\
 serveraction powerstatus
/opt/dell/srvadmin/sbin/racadm -r <DRAC interface IP address> -u <bmcusername> -p <bmcpassword>\
 serveraction hardreset
```

The BMC username/password values can be obtained from `cmsh` as follows:

```
[root@basecm11 ~]# cmsh
[basecm11]% partition use base
[basecm11->partition[base]]% bmcsettings
[basecm11->partition[base]->bmcsettings]% get password
12345
[basecm11->partition[base]->bmcsettings]% get username
tom
[basecm11->partition[base]->bmcsettings]%
```

Sometimes the `bright` user does not have the right privilege to get the correct values. The `racadm` commands then fail.

The `bright` user privilege can be raised using the following command:

```
/opt/dell/srvadmin/sbin/racadm -r <DRAC interface IP address> -u root -p <root password> set \
iDRAC.Users.4.Privilege 511
```

Here it is assumed that the BMC user has the username `bright`, a userID 4, and the privilege can be set to `511`.

3.7.5 Identification With A BMC

3.7.6 Identification At A Lower Level, Directly With `ipmitool`

Sometimes it is useful to identify a node using BMC commands. This can be done by, for example, blinking a light via a BMC command on the node:

Example

```
ipmitool -U <bmcusername> -P <bmcpassword> -H <host IP> chassis identify 1
```

The exact implementation may be vendor-dependent, and need not be an `ipmitool` command. Such commands can be scripted and run from CMDaemon.

The `ipmitool` command is provided by loading the `ipmitool` modules environment. The modules environment (section 2.2) is provided by the `cm-ipmitool` and the `cm-freeipmi` packages.

For testing without a BMC, the example script at `/cm/local/examples/cmd/bmc_identify` can be used if the environment variable `$CMD_HOSTNAME` is set. The logical structure of the script can be used as a basis for carrying out an identification task when a physical BMC is in place, by customizing the script and then placing the script in a directory for use.

To have such a custom BMC script run from CMDaemon, the `BMCIdentifyScript` advanced configuration directive (page 918) can be used.

The `ipmitool` command can also be used for troubleshooting during provisoning (page 319).

3.7.7 Identification At A Higher Level, With The `identify` Command Of Device Mode

In addition to using `ipmitool` directly, BCM provides the `identify` command in `device` mode as a convenient wrapper. It supports turning the identification LED on or off, as well as querying its current status:

Example

```
[basecm11->device]% identify -n node001 on
[basecm11->device]% identify -n node001 off
[basecm11->device]% identify -n node001 status
```

The standard node selection options (`-n`, `-c`, `-g`, `-r`) can be used to target specific nodes.

3.7.8 BMC Event Log

The `bmceventlog` command in `device` mode retrieves the BMC event log for one or more nodes using Redfish. This is useful for diagnosing hardware issues such as fan failures, power events, and sensor threshold violations.

Example

```
[basecm11->device]% bmceventlog -n node001
Device   id  message                                    severity  timestamp
-------- --- ------------------------------------------ --------- -------------------------
node001  6   Host Powered ON.                           OK        2025-11-17T21:28:32+00:00
node001  401 Sensor 'Chassis_0_FAN_2_REAR' reading      Critical  2026-01-14T03:07:36+00:00
             of 0.000000 (RPMS) is below the
             2204.000000 lower critical threshold.
```

Options to `bmceventlog` include:

- `--tail <count>`: Show only the last `<count>` events. The default is 100.
- `--clear`: Clear the BMC event log.
- `-v, --verbose`: Show additional details.
- `--debug`: Enable debug output.

3.7.9 BMC Reset

The `bmcreset` command in `device` mode performs a hardware reset of the BMC controller on one or more nodes. This can be useful when the BMC becomes unresponsive or needs to be restarted after a configuration change.

Example

```
[basecm11->device]% bmcreset -n node001
```

Resetting the BMC temporarily disrupts out-of-band management access to the node. The BMC typically takes a few minutes to become available again after a reset. Power state and the running OS on the node are not affected.

3.7.10 BMC Port Queries

BMC Port Accessibility

The netcat utility can be used to detect if the default UDP port 623 for standard IPMI services on a network is open. For a service on IP address 10.184.154.15 the following session query can be carried out:

Example

```
nc -vz -u 10.184.154.15 623
Connection to 10.184.154.15 623 port [udp/asf-rmcp] succeeded!
```

The `asf-rmcp` part indicates the Alert Standard Format Remote Management And Control Protocol over UDP.

BMC Version

The IPMI version can often be discovered using the `nmap` port scanner with the `ipmi-version` LUA script:

Example

```
nmap -sU --script ipmi-version -p 623 10.184.154.15
```

3.7.11 Leak Detection

For liquid-cooled systems some leak detection commands may be available:

- the `leakdetection` command in `device` mode enables or disables leak detection on nodes. Leak detection monitors coolant leak sensors using the BMC.
- the `leakinfo` command in `device` mode displays detected leaks.
- The `leakdetectionoverview` command (section 3.3.2 of the *NVIDIA Mission Control Manual*) is a rack-centric approach to leak detection, available in the NVIDIA Mission Control edition of BCM. Basic `rack` mode is described in section 3.16 of the *Administration Manual*.

Enabling leakdetection on a node001, and disabling leakdetection for a rack `rack01` could be carried out with:

Example

```
[basecm11->device]% leakdetection enable -n node001
[basecm11->device]% leakdetection disable -r rack01
```

Options to `leakdetection` include:

- `enable`: Enable leak detection on the specified devices.
- `disable`: Disable leak detection on the specified devices.
- `-m, --message <message>`: Add an information message that is displayed in the device status.

The standard node selection options (`-n`, `-c`, `-g`, `-r`) can be used to target devices. Rack-level selection with `-r` is convenient for enabling or disabling leak detection across all nodes in a rack.

A device with leak detection disabled shows `leak:disabled` appended to its status line. This is not for devices that lack leak-detection hardware. It is for devices that have leak-detection hardware whose sensor is broken or is reporting a false positive: in either case the administrator can disable leak detection on that device to prevent an unwanted power-off until the sensor is replaced.

Example

```
[basecm11->device[node001]]% leakdetection disable
disabled leak detection node001

Thu Aug 13 09:36:49 2026 [notice] head: node001 [   UP   ], leak:disabled

[basecm11->device[node001]]% leakdetection enable
enabled leak detection node001

Thu Aug 13 09:36:56 2026 [notice] head: node001 [   UP   ]
```

A device with leak detection enabled, which is the default, shows no `leak:` suffix on its status line. Typically, and ideally, running the `leakinfo` command shows nothing of interest:

Example

```
[basecm11->device]% leakinfo
Hostname         Timestamp Severity  Info
---------------- --------- --------- ----------------
[basecm11->device]%
```

If there are issues on the devices, then the output may look similar to::

```
[basecm11->device]% leakinfo
Hostname        Timestamp Severity  Info
--------------- --------- --------- ----------------------------------------
a05-dgx-01-c09  15:09:23  large     The state of resource `Chassis_0_LeakDete...
a07-dgx-03-c16  15:38:48  large     The state of resource `Chassis_0_LeakDete...
b05-dgx-05-c08  09:11:06  large     The state of resource `Chassis_0_LeakDete...
b05-dgx-05-c10  09:11:05  large     The state of resource `Chassis_0_LeakDete...
b06-dgx-06-c17  11:49:30  large     The state of resource `Chassis_0_LeakDete...
[basecm11->device]%
```

3.8 Configuring Switches And PDUs

3.8.1 Configuring With The Manufacturer's Configuration Interface

Network switches and PDUs that are to be used as part of the cluster should be configured with the PDU/switch configuration interface described in the PDU/switch documentation supplied by the manufacturer. Typically the interface is accessed by connecting using a web browser or telnet to an IP address preset by the manufacturer.

The IP address settings of the PDU/switch must match the settings of the device as stored by the cluster manager.

- In Base View, this is done using the navigation path `Devices > All Devices >` <*switchname*> to select the switch. If the switch does not already exist, then it can be added using the `+ Add` button. The values in the associated `Settings` window that comes up (figure 3.11) can then be filled in, and the IP address can be set and saved.
- In `cmsh` this can be done in `device` mode, with a `set` command:

  Example

  ```
  [root@basecm11 ~]# cmsh
  [basecm11]% device
  [basecm11->device]% set switch01 ip 10.141.253.2
  [basecm11->device*]% commit
  ```

Port Assignments For Switches

- Using the `uplinks` option to configure uplink ports is described in section 3.9.3.
- The `showport` tool for seeing what MAC address matches what port number on a switch is described in section 3.9.4.
- The `switchoverview` tool gives an overview of the MAC addresses detected on the ports based on SNMP queries, and is described in section 3.9.6.
- The `switchports` command lists the switches and switch ports that have been assigned a node.
- The `switchports` option assigns ports on a switch to a node:

  Example

  ```
  [basecm11->device]% #the option:
  [basecm11->device]% set node003 switchports switch01:1
  [basecm11->device*]% commit
  [basecm11->device]% #the command:
  [basecm11->device*]% switchports
  Switch           #Port  Node
  ---------------- ------ ----------------
  switch01         1      node003
  ```

The switchports option used at node level is a legacy option, which can still be used in BCM version 10. However it may not work as expected when a node has multiple interfaces that may be connected to a switch.

Since BCM version 10 it is therefore recommended to assign ports from within the interfaces submode for a node. This assigns ports clearly per interface, and supports bonding of the interfaces (section 3.5). Management of ports at interface level also makes it possible for CMDaemon Lite (section 2.6.7) to manage the network configuration of Cumulus switches (section 3.9).

Example

```
[basecm11->device]% use node003; interfaces
[basecm11->device[node003]->interfaces]% set eth1 switchports switch01:2
[basecm11->device*[node003*]->interfaces*]% set eth2 switchports switch01:3
[basecm11->device*[node003*]->interfaces*]% commit; ..
[basecm11->device[node003]]% switchports
Switch           #Port  Node
---------------- ------ ----------------
switch01         1      node003
switch01         2      node003
switch01         3      node003
```

Managing PDUs in Base View or cmsh is done in a similar way to the preceding method for switches. However, assigning PDUs and PDU ports to devices such as nodes is not part of this method, and is instead described in section 4.1.1.

APC PDUs

For the APC brand of PDUs, the powercontrol value for the PDU device should be set to apc.

For example, for a PDU with the name mypdu, its value can be set in cmsh with:

Example

```
[basecm11->device[mypdu]]% set powercontrol apc; commit
```

and in Base View with the navigation path:

Devices > All Devices > Power Distribution Units > Power Control

If it is not set to apc, then the list of PDU ports is ignored by default.

3.8.2 Configuring SNMP

BCM can use SNMP to manage devices such as rack sensors, power distribution units (PDUs), and switches (including many InfiniBand switches). This requires that SNMP be enabled. If disablesnmp is set to no, the default, then SNMP is enabled. For example for a switch:

Example

```
[basecm11]% device add switch myibswitch
[basecm11->device[myibswitch]]% get disablesnmp
no
```

Configuring SNMP Community Strings

In order to allow the cluster management software to communicate with a device that is managed using SNMP, SNMP must be enabled on it, and the SNMP community strings should be configured correctly.

By default, the SNMP community strings for switches and PDUs are set to public and private for read and write access respectively. If different SNMP community strings have been configured in the switch or PDU, the readstring and writestring properties of the corresponding switch device should be changed.

Example

```
[basecm11]% device use switch01
[basecm11->device[switch01]]% snmpsettings
[basecm11->device[switch01]->snmpsettings]% get readstring
public
[basecm11->device[switch01]->snmpsettings]% get writestring
private
[basecm11->device[switch01]->snmpsettings]% set readstring public2
[basecm11->device*[switch01*]->snmpsettings*]% set writestring private2
[basecm11->device*[switch01*]->snmpsettings*]% commit
```

Alternatively, these properties can also be set in Base View using the navigation path:

`Devices` > `All Devices` > `Switch` > > `SNMP Settings` >

Configuring SNMP Settings

SNMP settings can be configured in `cmsh` via the `snmpsettings` submode, which is available under `partition` mode as well as under `device` mode. The `partition` mode values are inherited by default by devices using SNMP settings at node level.

The effective (inherited) values can be displayed without entering the submode by using the `-s|--show` option, for example `snmpsettings -s` (section 3.7.2).

The submode allows the version to be set to `v1`, `v2c`, or `v3`.

Setting the version to the value `file` is also an option, but is not meant as an option for end users. It is used in SNMP walk emulation for debugging non-standard switches.

The SNMPv3 settings that can be managed in BCM are:

Example

```
[basecm11]% device use switch01
[basecm11->device[switch01]]% snmpsettings
[basecm11->device[switch01]->snmpsettings]% show
Parameter                      Value
------------------------------ ------------------------------------------------
Authentication key             < not set >
Authentication protocol        MD5
Context
Privacy key                    < not set >
Privacy protocol               DES
Retries                        -1
Revision
Security level                 Authentication encrypted
Security name
Timeout                        0s
VLAN Timeout                   0s
version                        V3
```

The `set` command can be used, sometimes with tab-completion, to set the SNMP switch parameters. For example, for the SNMPv3 parameters that are set to use cryptographic keys:

Example

```
[basecm11->device*[switch01*]->snmpsettings*]% set authenticationprotocol <TAB><TAB>
md5  sha
[basecm11->device*[switch01*]->snmpsettings*]% set authenticationprotocol sha
```

```
[basecm11->device*[switch01*]->snmpsettings*]% set privacyprotocol <TAB><TAB>
aes  des
[basecm11->device*[switch01*]->snmpsettings*]% set privacyprotocol aes
[basecm11->device*[switch01*]->snmpsettings*]% commit
[basecm11->device[switch01]->snmpsettings]%
```

SNMP Traps

BCM can assign an SNMP trap manager role to a node. The `snmptrapd` daemon is then configured and managed on the assigned node by CMDaemon.

Configuration options include enabling or disabling mailing of the messages, and setting the sender and recipients for the mail. By default an undefined value for `Server` means that the SNMP server is `localhost`.

Example

```
[root@basecm11 ~]# cmsh
[basecm11]% device use node001
[basecm11->device[node001]]% roles
[basecm11->device[node001]->roles]% assign snmptrap
[basecm11->device[node001]->roles*[snmptrap*]]% show
Parameter                      Value
------------------------------ ------------------------------------------------
Name                           snmptrap
Revision
Type                           SnmpTrapRole
Add services                   yes
Event                          yes
Mail                           yes
Recipients
All administrators             no
Access                         public
Server
Sender
Arguments
Alternative script
```

3.9 Configuring Cumulus Switches

A Cumulus switch is a switch that runs Cumulus Linux, which is a Debian-based distribution with a networking focus.

The following capabilities and features are available, or can be run, on a switch that can run Cumulus Linux:

- ONIE (Open Network Install Environment, `https://opencomputeproject.github.io/onie/`): a bootloader, similar in concept to PXE booting. It allows the switch to boot up and install the Cumulus OS from an image on the network. Installing Cumulus with ONIE wipes out any existing image already installed on the switch.
- ZTP (Zero Touch Provisioning): a protocol that is used with the `ztp` client. The protocol uses specially defined DHCP options. ZTP allows the switch to automatically carry out provisioning for other hardware devices on a network on top of an OS on the switch. The capability to carry this out becomes available automatically when the switch is powered on, and the interface on which provisioning is to be carried out automatically becomes active with an IP address ("day-0 provisioning").

- NVUE (NVIDIA User Experience): a CLI that uses the `nv` command set to manage the network configuration of the switch. The NVUE CLI is documented at `https://docs.nvidia.com/networking-ethernet-software/cumulus-linux-55/System-Configuration/NVIDIA-User-Experience-NVUE/NVUE-CLI/`.
- The NVUE REST API: if it is explicitly enabled, can run the same commands via HTTP as the NVUE CLI. The NVUE REST API is documented at `https://docs.nvidia.com/networking-ethernet-software/cumulus-linux-55/System-Configuration/NVIDIA-User-Experience-NVUE/NVUE-API/`
- Public key SSH: if configured, the administrator can access the switch from the head node safely and securely with public key SSH, including passwordless SSH.

A Cumulus switch is added to the cluster manager as a regular switch object:

Example

```
[basecm11]% device add switch myswitch
[basecm11->device*[myswitch*]]% # hasclientdaemon: set if CMDaemon Lite should run on switch
[basecm11->device*[myswitch*]]% set hasclientdaemon yes
[basecm11->device*[myswitch*]]% interfaces
[basecm11->device*[myswitch*]->interfaces]% add physical eth0
[basecm11->device[myswitch*]->interfaces*[eth0*]]% set mac 12:34:56:78:90:AB
[basecm11->device[myswitch*]->interfaces*[eth0*]]% set ip 1.2.3.4
[basecm11->device[myswitch*]->interfaces*[eth0*]]% commit
[basecm11->device[myswitch]->interfaces[eth0]]% ..;..
[basecm11->device[myswitch]]% status
myswitch ................ [  DOWN  ]
```

Cumulus access settings (described later on, starting from page 150) and ZTP settings (described later on, starting from page 155) must also be configured for the switch to work correctly.

Committing the Cumulus switch object configuration automatically configures the DHCP and the DNS services. The DHCP server is configured by CMDaemon to use the ZTP custom provisioning script. The ZTP provisioning script is created on-demand, along with a directory, from a default template (page 157) when the switch boots. The creation can alternatively be forced right away by running the `initialize` command.

3.9.1 Cumulus Switches Access Configuration, Initialization And Network Device Discovery

Cumulus Switches Access Configuration

By default, Cumulus switches use certificate-based authentication for CMDaemon access, just like regular nodes. DGX SuperPODS are configured in that manner.

However, CMDaemon Lite (section 2.6.7) is recommended instead of CMDaemon for Cumulus switches on most other clusters. If CMDaemon Lite is to run on the switch, then the `hasclientdaemon` parameter must be enabled for it:

Example

```
[basecm11]% device use mycumulus
[basecm11->device*[mycumulus*]]% set hasclientdaemon yes
[basecm11->device*[mycumulus*]]% commit
```

The cluster administrator must configure a username/password pair for SSH public key authentication with CMDaemon Lite.

- Since BCM version 11, Cumulus switch access via a username/password pair is configured in the `accesssettings` submode. Updates can be done using ZTP or NV.

- In earlier versions of BCM SNMP settings were used to set the username/password pair.

Example

```
[basecm11->device]% add switch mycumulus
[basecm11->device*[mycumulus*]]% accesssettings
[basecm11->device*[mycumulus*]->accesssettings*]% set username cumulus; set password 1234
[basecm11->device*[mycumulus*]->accesssettings*]% show
Parameter                      Value
------------------------------ ------------------------------------------------
Revision
Username                       cumulus
Password                       *********
Rest port                      8765
Update in ztp                  no
Update in NV                   no
[basecm11->device*[mycumulus*]->accesssettings*]% commit
```

If all switches are Cumulus switches, then setting the username/password can be carried out at `partition` level with:

Example

```
[basecm11->partition[base]]% accesssettings
[basecm11->partition[base]]->accesssettings]% set username cumulus; set password 1234
[basecm11->partition*[base*]]->accesssettings*]% commit
```

The effective (inherited) values for a device, category, or partition can be displayed without entering the submode by using the `-s|--show` option, for example `accesssettings -s` (section 3.7.2).

Compared with a regular node (section 6.2), a Cumulus switch has these differences when setting a username and password:

- The password change only takes effect on the switch after the switch reboots.
- If the `startup.yaml` file used by NV does not set the password, and `cm-lite-daemon` is not installed, then during reboot either
  - `Update in ztp` must be set to `yes` to allow the password change using ZTP
    or
  - `Update in NV` must be set to `yes`, to allow the password change using NV

The REST API port, with a default value of `8765`, can also be modified from the `accesssettings` submode, to match the deployment specifications.

The effective Access (and BMC) credentials for any subset of devices, including the inheritance source (device, category, or partition), can be reviewed with the `credentialsoverview` command described in section 3.7.3.

Other parameters for CMDaemon Lite integration are described in the section starting on page 152, **Custom Services Option 1: Cumulus With CMDaemon Lite**. Also covered in that section, on page 155, are the ZTP settings. ZTP settings are managed via the ZTP mode of `cmsh`, and are needed for the switch to pick up its ZTP provisioning script.

The Cumulus Custom Discovery Script (Deprecated)

This is a legacy script that comes with BCM, and is at `/cm/local/apps/cmd/scripts/cm-cumulus-switch.py`.

Setting the script and running the `initialize` command for the Cumulus switch in `device` mode initializes access settings and carries out network discovery for the Cumulus switch, in versions of BCM prior to version 10.

Since BCM version 10, the script is only needed for the Cumulus Linux version 4 series. Since Cumulus Linux version 5, and since BCM version 10, the script is no longer required, due to the integration of Cumulus with BCM.

A Cumulus switch running version 4 of Cumulus Linux can be set up and initialized on BCM version 10 with:

Example

```
[basecm11->device*[mycumulus*]]% set controlscript /cm/local/apps/cmd/scripts/cm-cumulus-switch.py
[basecm11->device*[mycumulus*]]% commit
[basecm11->device[mycumulus]]% initialize
```

3.9.2 Custom Service Setups For Cumulus Linux

There are two custom service setups that are supported for cluster management on top of Cumulus Linux:

1. a setup that uses CMDaemon Lite (page 152, **Custom Services Option 1: Cumulus With CMDaemon Lite**)
2. a setup that uses YAML to configure the services (page 157, **Custom Services Option 2: Cumulus With A YAML Preconfiguration**).

These can actually be run together, if the services do not conflict with each other. For example, the `nvued` service can be installed via YAML, to provide the NVUE, independently of CMDaemon Lite. It is however useful to describe these custom service setups individually.

Custom Services Option 1: Cumulus With CMDaemon Lite

If the Cumulus switch is defined and configured with CMDaemon Lite, then this allows some Cumulus features to be managed via BCM, as well as some of the standard features of BCM to work on the switch. The following features can be managed:

- monitoring (Chapter 10). For example, the latest data values (section 10.6.3) of the switch, including the bytes going through the many ports, can be seen with:

  Example

```
[basecm11->device[myswitch]]% latestmonitoringdata
Measurable       Parameter       Type          Value          Age       State     Info
---------------- --------------- ------------ ------------- --------- --------- ---------
AlertLevel       count           Internal     0             1m 5s
AlertLevel       maximum         Internal     0             1m 5s
AlertLevel       sum             Internal     0             1m 5s
BufferMemory                     Memory       113 MiB       2m 5s
BytesRecv        eth0            Network      22.3917 B/s   2m 5s
BytesRecv        mgmt            Network      6.79167 B/s   2m 5s
BytesRecv        mirror          Network      0 B/s         2m 5s
BytesRecv        swid0_eth       Network      0 B/s         2m 5s
BytesRecv        swp1            Network      0 B/s         2m 5s
BytesRecv        swp2            Network      0 B/s         2m 5s
```

```
BytesRecv        swp3            Network      0 B/s         2m 5s
BytesRecv        swp4            Network      0 B/s         2m 5s
...
```

- system information can be viewed with the `sysinfo` command:

  Example

```
[basecm11->device[myswitch]]% sysinfo
Name                      Value
------------------------- ------------------------------------------------------------
BIOS Version              1.3
BIOS Vendor               American Megatrends Inc.
BIOS Date
Motherboard Manufacturer
Motherboard Name
System Manufacturer       NVIDIA
System Name               SN2201
Vendor Tag
Total Memory              8048771072 bytes (7.496GB)
Swap Memory               0 bytes (0B)
OS Name                   Linux
OS Version                5.10.0-cl-1-amd64
OS Flavor                 #1 SMP Debian 5.10.162-1+cl5.4.0u1 (2023-01-20)
Number of Physical CPUs   1
Number of Cores           2
Core 0-1                  Intel(R) Atom(TM) CPU C3338R @ 1.80GHz
Number of Disks           1
Total Disk Space          115,923,419,136 bytes (115GB)
Disk /dev/nvme0n1p4       (19,320,569,856 bytes, 19.3GB) ()
SELinux                   no
FIPS                      no
Fabric                    no
Age                       21h 58m
ZTP/date                  Tue Apr 11 09:50:21 2023 UTC
ZTP/method                ZTP DHCP
ZTP/result                success
ZTP/state                 enabled
ZTP/url                   http://10.141.255.254:8080/switch/myswitch/cumulus-ztp.sh
ZTP/version               1.0
[basecm11->device[myswitch]]%
```

- systemd services can be added for the device via roles, and listed in the `services` submode (section 3.15). Management of services is a standard part of CMDaemon Lite since NVIDIA Base Command Manager version 10.23.06.

- configuration via `cmsh`: can be set up via either a YAML file, or manually, or automatically with some optional manual parts. The appropriate mode is set with `nvconfigurationmode` for the Cumulus device:

```
[basecm11->device[cumulus02]]% set nvconfigurationmode <tab><tab>
auto    file    manual
```

  – `auto`: BCM settings such as for `hostname` or `timezone` for a Cumulus switch network configuration are converted automatically to run as `nv` commands. All the `nv` commands carried out

on the switch via BCM, whether from an automatic conversion or not, can be viewed within nvconfiguration mode. The nv commands that have been automatically converted from BCM settings are assigned a type value of auto, and can be viewed within nvconfiguration mode:

Example

```
[basecm11->device[myswitch]->nvconfiguration]% show
Type     #Index   Command
-------- -------- ------------------------------------------------------------
auto     1        nv set system hostname myswitch
auto     2        nv set system timezone Europe/Amsterdam
auto     3        nv set service snmp-server enable on
auto     4        nv set service snmp-server listening-address all
auto     5        nv set service snmp-server listening-address all-v6
auto     6        nv set service snmp-server readonly-community public access any
auto     7        nv set service ntp mgmt pool 10.141.255.254
auto     8        nv set service dns mgmt server 10.141.255.254
auto     9        nv set service syslog mgmt server 10.141.255.254 port 514
auto     10       nv set service syslog mgmt server 10.141.255.254 protocol udp
auto     11       nv set bridge domain br_default type vlan-aware
[basecm11->device[myswitch]->nvconfiguration]%
```

The settings are applied to the switch when the apply command is run within nvconfiguration mode.

– manual: If nvconfigurationmode is set to manual then the commands of type auto (as shown within the preceding example) are ignored. Direct nv commands can be entered by the administrator manually from within the manual NV Configuration mode.

  Direct nv commands can actually also be added within auto mode for nvconfiguration, and are in that case also of type manual:

Example

```
[basecm11->device[myswitch]->nvconfiguration]% nv set interface swp27 ip address 10.141.255.123
[basecm11->device*[myswitch*]->nvconfiguration*]% show
Type     #Index   Command
-------- -------- ------------------------------------------------------------
auto     1        nv set system hostname myswitch
...
auto     11       nv set bridge domain br_default type vlan-aware
manual   1        nv set interface swp27 ip address 10.141.255.123
```

The changes are applied when the apply command is run within nvconfiguration mode.

Only commands of type manual can be removed from nvconfiguration mode. The removal can be carried out from within nvconfiguration mode with the help of the manual command and the nv del command:

Example

```
[basecm11->device*[myswitch]->nvconfiguration]%  ..; get nvconfigurationmode; nvconfiguration
AUTO
[basecm11->device*[myswitch]->nvconfiguration]% show
...
auto     11       nv set bridge domain br_default type vlan-aware
manual   1        nv set interface swp27 ip address 10.141.255.123
manual   2        nv set system timezone Europe/Berlin
```

```
manual   3        nv set service snmp-server listening-address all-v6

[basecm11->device*[myswitch]->nvconfiguration]% manual   #nvconfiguration mode changes to manual
                                                         #copies all types to be manual
[basecm11->device*[myswitch]->nvconfiguration]%  ..; get nvconfigurationmode; nvconfiguration
MANUAL
[basecm11->device*[myswitch]->nvconfiguration]% show
manual   1        nv set interface swp27 ip address 10.141.255.123
manual   2        nv set system timezone Europe/Berlin
manual   3        nv set service snmp-server listening-address all-v6
[basecm11->device*[myswitch]->nvconfiguration]% nv del 1-2
[basecm11->device*[myswitch]->nvconfiguration]% commit
[basecm11->device*[myswitch]->nvconfiguration]% apply  #only now is switch updated
```

The `manual` command switches `nvconfigurationmode` to `manual`, and copies the stack of `auto` type commands over to the stack in `manual` mode. The `commit` command saves the configuration in the CMDaemon database. As usual, the configuration defined by the `manual` NV configuration mode is only applied to the switch after using the `apply` command.

The stack in the `manual` mode stack is then what is used on the switch instead of the `auto` mode stack, as per the CMDaemon state.

– `file`: If `nvconfigurationmode` is set to `file`, then a YAML file is used by BCM to carry out the commands. A sample YAML file that uses the currently running configuration of the switch can be displayed by running
`nv config show`
on the switch.

Example

```
[basecm11->device[myswitch]]% set nvconfigurationmode file
[basecm11->device*[myswitch*]]% set nvconfigurationfile startup.yaml
[basecm11->device*[myswitch*]]% commit
```

The `apply` command is not used for `file` mode, since the commands are carried out over ZTP.

- the ZTP configuration can be managed from `ztpsettings` mode:

Example

```
[basecm11->device[cumulus02]]% ztpsettings
[basecm11->device[cumulus02]->ztpsettings]% show
Parameter                          Value
---------------------------------- ------------------------------------------------
Revision
Script template                    cumulus-ztp.sh
JSON template
Image                              cumulus-linux-5.13.1-mlx-amd64.bin
Check image on boot                yes
Run ZTP on each boot               yes
Install lite daemon                yes
Authorized key file root           /root/.ssh/authorized_keys
Authorized key file cumulus
Authorized key file admin
Enable API                         no
Enable external access API         no
Merge key value settings partition  no
```

```
Key value settings                  <submode>
Firmwares
Preinstall scripts
Post-install scripts
PTM topology file
```

The effective (inherited) values can be displayed without entering the submode by using the `-s|--show` option, for example `ztpsettings -s` (section 3.7.2).

Notes about some of the `ztpsettings`:

- By default, `authorizedkeyfileroot` is not set, which means that user `root` cannot access the switch with public key authentication.
- By default, `authorizedkeyfilecumulus` is not set. The user `cumulus` is the API user with administrator privileges for the Cumulus switch.
- By default, `authorizedkeyfileadmin` is not set. The user `admin` is the API user with administrator privileges for the NVLink switch.
- Cumulus images for the switch, set by `image`, are provisioned from the head node. Cumulus images can be picked up from `https://enterprise-support.nvidia.com/s/downloads` and can then be placed on the head node at `/cm/local/apps/cmd/etc/htdocs/switch/images/`.
- The `checkimageonboot` setting checks that the existing image on the switch matches the image that can be offered via ZTP. If it does not, then the switch updates its image, picking it up via ZTP.
- The file specified by `scripttemplate` is used to generate a ZTP provisioning file on-demand when the switch boots. Running `initialize` from the switch object level (at `cumulus02` level in the preceding example) generates the file without a reboot.
- The file specified by `PTM topology file` describes the cabling topology.

- The `switchoverview` command returns output such as:

```
[basecm11->device[cumulus01]]% switchoverview
Device: cumulus01
State : [   UP   ]
Model : Cumulus Linux 5.2.1
Port    Name    Status  Assigned        Uplink Speed           Detected
----    ------  ------- ------------    ------ -----------     ----------------------------
1       swp1    DOWN                    no     0 b/s           bridge domains: cm-default
2       swp2    DOWN                    no     0 b/s           bridge domains: cm-default
3       swp3    DOWN                    no     0 b/s           bridge domains: cm-default
4       swp4    DOWN                    no     0 b/s           bridge domains: cm-default
5       swp5    UP      node001-dpu     no     100 Gb/s        bridge domains: cm-default
6       swp6    DOWN                    no     0 b/s           bridge domains: cm-default
7       swp7    UP      node002-dpu     no     100 Gb/s        bridge domains: cm-default
```

- The `switchports` command (page 146) is used to configure the port assignment between the nodes and the switch ports. Configuring the port assignment at `interfaces` level is needed to allow CMDaemon Lite to manage Cumulus network configuration.

Example

```
[basecm11->device[node002]]% interfaces node004
[basecm11->device[node004]->interfaces]% set DPU1 switchports cumulus01:4
```

Custom Services Option 2: Cumulus With A YAML Preconfiguration

A YAML configuration can be used outside of `cmsh`. DGX SuperPODS use this option during the standard BCM installation. It means that CMDaemon Lite running on the switch (section 3.9.2) does not need to be used. Instead, two custom-generated YAML files are used.

The YAML files that are generated are:

- `startup.yaml`: this contains a sequence of Cumulus `nv` commands
- `cm-startup.yaml`: this contains the BCM DGX/switch definitions to be added by pythoncm

The `cm-startup.yaml` file is parsed, and the devices that are defined within it are configured with the PythonCM (Chapter 1 of the *Developer Manual*) script, `cm-import-startup`.

The devices are then powered on in the correct order.

The YAML file `startup.yaml` can be handled manually (using `scp` and the `nv apply` command), or by using ZTP, or by using Ansible.

Settings Applied Via ZTP

The configuration script `cumulus-ztp.sh` template, found under `/cm/local/apps/cmd/etc/htdocs/switch/template`, can be customized to suit the requirements.

The YAML file `startup.yaml` is placed under:

`/cm/local/apps/cmd/etc/htdocs/switch/<`*switch or host name*`>/`

This allows it to be picked up automatically via ZTP, and the `nv` commands are applied it to the switch on boot.

CMDaemon Lite is not required, but can be installed separately.

Settings Applied Via Ansible

Alternatively, Ansible can be used to push the YAML configuration to each switch.

With an Ansible installation, ZTP is not required, but it can be used. CMDaemon Lite is not required for applying the settings via Ansible either, but can be installed separately.

3.9.3 Uplink Ports

Uplink ports are switch ports that are connected to other switches. CMDaemon must be told about any switch ports that are uplink ports, or the traffic passing through an uplink port will lead to mistakes in what CMDaemon knows about port and MAC correspondence. Uplink ports are thus ports that CMDaemon is told to ignore.

To inform CMDaemon about what ports are uplink ports, Base View or `cmsh` are used:

- In Base View, the switch is selected, and uplink port numbers can be added via the navigation path:
  `Devices > All Devices > Switch`  `> Uplinks > + Add`
  (figure 3.11):

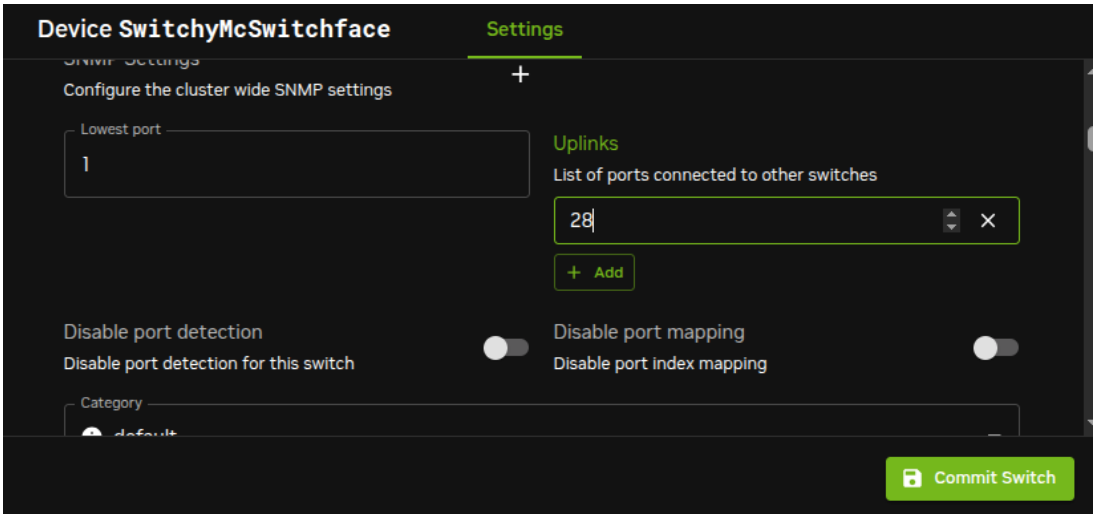


Figure 3.11: Notifying CMDaemon about uplinks with Base View

The state is saved with the `Commit Switch` button of figure 3.11.

- In `cmsh`, the switch is accessed from the `device` mode. The uplink port numbers can be appended one-by-one with the `append` command, or `set` in one go by using space-separated numbers.

Example

```
[root@basecm11 ~]# cmsh
[basecm11]% device
[basecm11->device]% set switch01 uplinks 15 16
[basecm11->device*]% set switch02 uplinks 01
[basecm11->device*]% commit
successfully committed 3 Devices
```

3.9.4 The `showport` MAC Address to Port Matching Tool

The `showport` command can be used in troubleshooting network topology issues, as well as checking and setting up new nodes (section 5.4.2).

Basic Use Of `showport`

In the `device` mode of `cmsh` is the `showport` command, which works out which ports on which switch are associated with a specified MAC address.

Example

```
[root@basecm11 ~]# cmsh
[basecm11]% device
[basecm11->device]% showport 00:30:48:30:73:92
switch01:12
```

When running `showport`, CMDaemon on the head node queries all switches until a match is found.

If a switch is also specified using the "`-s`" option, then the query is carried out for that switch first. Thus the preceding example can also be specified as:

```
[basecm11->device]% showport -s switch01 00:30:49.00:73:92
switch01:12
```

If there is no port number returned for the specified switch, then the scan continues on other switches.

Mapping All Port Connections In The Cluster With `showport`

A list indicating the port connections and switches for all connected devices that are up can be generated using this script:

Example

```
#!/bin/bash
for nodename in $(cmsh -c "device; foreach * (get hostname)")
do
  macad=$(cmsh -c "device use $nodename; get mac")
  echo -n "$macad $nodename "
  cmsh -c "device showport $macad"
done
```

The script may take a while to finish its run. It gives an output like:

Example

```
00:00:00:00:00:00 switch01: No ethernet switch found connected to this mac address
00:30:49.00:73:92 basecm11: switch01:12
00:26:6C:F2:AD:54 node001: switch01:1
00:00:00:00:00:00 node002: No ethernet switch found connected to this mac address
```

Related utilities to `showport` are:

- `finddevicebyswitchport` (section 3.9.7) which—as the name suggests—finds the devices for each switch port
- `setmacviaswitchport` (section 3.9.8), running in `--dry-run` mode, which shows the discovered MAC address for a switch

3.9.5 Disabling Port Detection

An administrator may wish to disable node identification based on port detection. For example, in the case of switches with buggy firmware, the administrator may feel more comfortable relying on MAC-based identification. Disabling port detection can be carried out by clearing the `switchports` setting of a node, a category, or a group. For example, in `cmsh`, for a node:

Example

```
[root@basecm11 ~]# cmsh
[basecm11]% device use node001
[basecm11->device[node001]]% clear switchports
[basecm11->device*[node001*]]% commit
[basecm11->device[node001]]%
```

Or, for example for the `default` category, with the help of the `foreach` command:

Example

```
[root@basecm11 ~]# cmsh
[basecm11]% device
[basecm11->device]]% foreach -c default (clear switchports); commit
```

3.9.6 The `switchoverview` Command

Also within `device` mode, the `switchoverview` command gives an overview of MAC addresses detected on the ports, and some related properties. The command works using SNMP queries. Output is similar to the following (some lines ellipsized):

```
[basecm11->device]% switchoverview dell-switch1
Device: dell-switch1
State : [   UP   ]
Model : 24G Ethernet Switch

Port Assignment:

Port   Status Assigned         Uplink Speed    Detected
------ ------ ---------------- ------ -------- -------------------------\
-----------------------------------------------------------------------
1      UP                      false  1 Gb/s   -
2      UP                      false  1 Gb/s   -
3      UP                      false  1 Gb/s   74:86:7A:AD:3F:2F, node3
4      UP                      false  1 Gb/s   74:86:7A:AD:43:E9, node4
...
11     UP                      false  1 Gb/s   74:86:7A:AD:44:D8, node11
12     UP                      false  1 Gb/s   74:86:7A:AD:6F:55, node12
...
23     UP                      false  1 Gb/s   74:86:7A:E9:3E:85, node23
24     UP                      false  1 Gb/s   74:86:7A:AD:56:DF, node24
49     UP                      false  10 Gb/s  74:86:7A:AD:68:FD, node1
50     UP                      false  10 Gb/s  74:86:7A:AD:41:A0, node2
53     UP     node34           false  1 Gb/s   5C:F9:DD:F5:79.0D, node34
54     UP     node35           false  1 Gb/s   5C:F9:DD:F5:45:AC, node35
...
179    UP                      false  1 Gb/s   24:B6:FD:F6:20:6F,\
 24:B6:FD:FA:64:2F, 74:86:7A:DF:7E:4C, 90:B1:1C:3F:3D:A9,\
 90:B1:1C:3F:51:D1, D0:67:E5:B7:64:0F, D0:67:E5:B7:61:20
180    UP                      false  100 Mb/s QDR-switch
205    UP                      true   10 Gb/s  -
206    DOWN                    false  10 Gb/s  -
...
[basecm11 ->device]%
```

3.9.7 The `finddevicebyswitchport` Command

The `finddevicebyswitchport` command, in `device` mode, is the converse of the `showport` command (section 3.9.4). Instead of finding the switch port for a known MAC address, it finds which device is connected to a specified switch port. The port is specified in the form:

<*switch name*>:<*port number*>

An optional breakout port can be specified with:

<*switch name*>:<*port number*>s<*breakout port number*>

and several ports can be queried in one command.

Example

```
[basecm11->device]% finddevicebyswitchport switch01:1
node001
[basecm11->device]% finddevicebyswitchport switch01:1 switch01:2
switch01:1                      node001
switch01:2                      node002
```

When only one port is queried, just the matching hostname is shown. When several ports are queried, each line is prefixed with the <*switch name*>:<*port number*> that was looked up. As with showport, the lookup relies on the port-to-MAC mapping that CMDaemon has discovered for the switch. If no device is associated with a specified port, then no hostname is returned for that port.

3.9.8 The setmacviaswitchport Command

The setmacviaswitchport command, in device mode, sets the MAC address of a device based on the MAC address discovered on the switch port to which the device is connected. This is convenient when provisioning hardware whose MAC address is not yet known to CMDaemon, but whose switch port connection is known.

The --dry-run option finds the MAC addresses, but does not save the change. Device selection options such as -t (type), -n (nodes), and -c (category), can be used to act on a set of devices. The -f|--force option forces an overwrite of devices that already have a MAC set, and the --index option selects which MAC to use, in case the switch port has more than one MAC associated with it.

Example

```
[basecm11->device]% setmacviaswitchport --dry-run -t switch
switch01     no         no force and MAC already set
switch02     no         no force and MAC already set
switch03     yes        00:00:00:00:00:01
...
```

The second column indicates whether the MAC was (or, for a dry run, would be) saved, while the last column gives the discovered MAC address, or the reason that no change was made. Running the command without the --dry-run option commits the discovered MAC addresses.

3.9.9 The checkfactorydefaultpassword Command

The checkfactorydefaultpassword command, in device mode, checks whether switches still have their factory default password set, by attempting to authenticate against them. With the --update option, it also changes the factory password to the target password that is defined in the accesssettings submode of the device. Switches can be selected by name, by a range of names, or with device selection options such as -t switch.

Example

```
[basecm11->device]% checkfactorydefaultpassword switch01..switch04
[basecm11->device]% checkfactorydefaultpassword --update switch01..switch04
[basecm11->device]% checkfactorydefaultpassword -t switch
```

Without the --update option the command only reports the status, and does not modify any passwords.

3.10 Additional device Mode Node Commands And Submodes

Besides the switch-oriented commands of the preceding section, device mode provides several commands and submodes for inspecting and configuring devices, including nodes.

3.10.1 The nodeoverview Command

The nodeoverview command displays a brief summary for a node. This includes its uptime, load averages, memory and workload manager (WLM) slot usage, per-interface network throughput, and per-mount disk usage. If no node is specified, then the current node is used.

Example

```
[basecm11->device]% nodeoverview node001
Device             node001
Uptime             3w 3d
Load               0.16 / 0.18 / 0.17
Memory used        64 GiB (3.19%)
Memory unused      1.93 TiB (98.4%)
Memory total       1.96 TiB
WLM slots used     0 (0.0%)
WLM slots unused   144 (100.0%)
WLM slots total    144

enp22s22f0np0  1.69 GiB     3.3 GiB
ibp2p3s0       0 B          0 B
...

/              21.1 GiB     1.60 TiB     98.7%
```

3.10.2 The additionalnodeinformation Command

The additionalnodeinformation command is run on a set of nodes, and displays administrator-registered information commands for those nodes. The set of registered commands can be listed with the --list option. The -a|--run-on-active-head-node option runs the command on the active head node instead of on the node itself. Nodes are selected with the usual selection options, such as -n, -c, or -r.

Example

```
[basecm11->device]% additionalnodeinformation -n node001 --list
info-1
info-2
```

Here info-1 and info-2 are the names of administrator-registered information commands. If no commands have been registered, then --list returns nothing.

Setting up the commands can be done with the cm-manipulate-advanced-config.py command (section C, page 925) as follows:

Example

```
[root@basecm11 ~]# cm-manipulate-advanced-config.py -g "AdditionalNodeInfo=info-1,info-2"
[root@basecm11 ~]# cm-manipulate-advanced-config.py -g "info-1=/cm/shared/info-1.sh"
[root@basecm11 ~]# cm-manipulate-advanced-config.py -g "info-2=/cm/shared/info-2.sh"
```

and then restarting cmd with systemctl restart cmd.

An example of a script for info-1.sh is:

Example

```
[root@basecm11 ~]# cat /cm/shared/info-1.sh
#!/bin/bash
/cm/local/apps/cmd/bin/cmsh -c "ds"
```

That script can then be run with:

Example

```
[basecm11->device]% additionalnodeinformation -n basecm11 info-1
name: info-1
path: /cm/shared/info-1.sh
exit code: 0
--- stdout ---
node001 .................. [   UP   ]
node002 .................. [   UP   ]
node003 .................. [   UP   ]
node004 .................. [   UP   ]
node005 .................. [   UP   ]
basecm11 ................. [   UP   ]
```

3.10.3 The minimalconfig Command

The minimalconfig command shows the minimal set of configuration objects that apply to a node. This includes the category, configuration overlays, head nodes, monitoring data producers, and other entities that make up a node configuration. It is useful when troubleshooting why a particular setting or monitoring item for a node may not work. If no node is specified, then the current node is used.

Example

```
[basecm11->device]% minimalconfig node001
Category                                    default
Configuration overlay                       slurm-client
Configuration overlay                       slurm-server
Head node                                   head01
Head node                                   head02
MonitoringDataProducerCPU                   CPUSampler
MonitoringDataProducerGPU                   GPUSampler
MonitoringDataProducerDeviceState           DeviceState
...
```

3.10.4 The checkremotemount Command

The checkremotemount command checks the status of remote mounts on a set of nodes. The mount to check is given as a path by default. The --file option evaluates the path as a file instead of a directory, the -f|--force option forces an update of the cached mounts data, and the --clear option clears the mount cache data. Nodes are selected with the usual selection options.

Example

```
[basecm11->device]% checkremotemount -n node001 /cm/shared
node001  1        good
```

The columns show the node, the number of matching mounts found, and the status of the mount.

3.10.5 The proxysettings Submode

The proxysettings submode, within device mode, configures the HTTP, HTTPS, and FTP proxy servers that a node uses, along with the list of hosts that should bypass the proxy.

Example

```
[basecm11->device]% proxysettings node001
[basecm11->device[node001]->proxysettings]% show
```

```
Parameter                      Value
------------------------------ ----------------------------------------------
HTTP proxy address
HTTP proxy username
HTTP proxy password
HTTPS proxy address
HTTPS proxy username
HTTPS proxy password
FTP proxy address
FTP proxy username
FTP proxy password
Hosts without proxy
```

3.10.6 The `selinuxsettings` Submode

The `selinuxsettings` submode, within `device` mode, configures the SELinux behavior for a node, such as the SELinux mode and policy, whether to initialize SELinux, whether to reboot after a context restore, and the context actions to apply for the various install scenarios.

Example

```
[basecm11->device]% selinuxsettings node001
[basecm11->device[node001]->selinuxsettings]% show
Parameter                      Value
------------------------------ ----------------------------------------------
Initialize                     yes
Reboot after context restore   no
Allow NFS home directories     yes
Context action auto install    auto
Context action full install    auto
Context action nosync install  always
Mode                           permissive
Policy                         targeted
Key value settings             <submode>
```

SELinux is typically carried out right after a clean installation (Chapter 12 of the *Installation Manual*).

3.10.7 The `remotedaemon` Command

The `remotedaemon` command, in `device` mode, installs or removes the CMDaemon package on a device that already has an operating system running, without going through a BCM software-image provisioning cycle. This is intended for devices that were brought up by something other than BCM and then registered with it. A DPU that has already been installed by DPF is the primary example. DPF installs the DPU's operating system independently of BCM, and `remotedaemon install` then deploys CMDaemon onto the already-running DPU, so that it can be managed and monitored by BCM. This is the same idea as `cm-lite-daemon` for switches, where the device's own operating system is left in place and only the BCM daemon is added to it.

The device must already be defined in BCM before `remotedaemon` is used. The command does not create the device object.

Example

```
[basecm11->device]% remotedaemon install dpu001
[basecm11->device]% remotedaemon remove dpu001
```

Options to `remotedaemon` include:

- `install`: Install the CMDaemon package on the specified devices. The `-f|--force` option reinstalls CMDaemon even if it is already present.
- `remove`: Remove the CMDaemon package from the specified devices. The `--purge-repo` option also removes the BCM package repository configuration from the device.

Prerequisites for `remotedaemon install` are:

- The target device is already defined in BCM as a compute node.
- The target is running Ubuntu, and is reachable over SSH from the active head node using the device access settings (username and password) configured for it.
- The target can reach the active head node over HTTPS, to fetch packages and its node certificate.

3.11 Configuring The NetQ Network Management System

NetQ telemetry data values are produced by NetQ. A running NetQ server can be integrated with BCM by configuring its credentials and connectivity settings in the `netqsettings` submode from within the `partition` mode of BCM:

Example

```
[basecm11->partition[base]]% netqsettings
[basecm11->partition*[base*]->netqsettings*]% show
Parameter                      Value
------------------------------ ------------------------------------------------
Revision
Server
User name
Password                       < not set >
Port                           443
Verify SSL                     no
CA certificate
Certificate
Private key
Prometheus metric forwarders   <0 in submode>
[basecm11->partition*[base*]->netqsettings*]% set username <NetQ username>
[basecm11->partition*[base*]->netqsettings*]% set password <NetQ password>
[basecm11->partition*[base*]->netqsettings*]% set server <NetQ hostname or IP address>
[basecm11->partition*[base*]->netqsettings*]% commit
```

The corresponding Base View navigation path for this is using:
`Cluster > Settings > BMC Settings > NetQ > +`

With Kubernetes, NetQ can also be configured with the NetQ operator (section 6.9 of the *Containerization Manual*).

Connecting to the NetQ server allows NetQ telemetry data values to be picked up by BCM. These values are then treated as measurables (section 10.2.1). NetQ measurables (sections G.1.13 and G.2.4) can be managed and displayed with BCM just like other measurables.

3.12 Configuring UFM

UFM, the NVIDIA Unified Fabric Manager, is a fabric manager, that handles routing on the fabric. An optional component, UFM Host Agent, allows performance to be monitored. A running UFM server can be integrated with BCM by configuring its credentials and connectivity settings in the `ufmsettings` submode from within the `partition` mode of BCM. This is very similar to how NetQ settings (section 3.11) are managed:

Example

```
[basecm11->partition[base]]% ufmsettings
[basecm11->partition*[base*]->ufmsettings*]% show
Parameter                      Value
------------------------------ ------------------------------------------------
Revision
Server
User name
Password                       < not set >
Port                           443
Verify SSL                     no
CA certificate
Certificate
Private key
Prometheus metric forwarders   <0 in submode>
[basecm11->partition*[base*]->ufmsettings*]% set username <UFM username>
[basecm11->partition*[base*]->ufmsettings*]% set password <UFM password>
[basecm11->partition*[base*]->ufmsettings*]% set server <UFM hostname or IP address>
[basecm11->partition*[base*]->ufmsettings*]% commit
```

The corresponding Base View navigation path for this is using:
`Cluster > Settings > BMC Settings > UFM settings> +`

3.13 Disk Layouts: Disked, Semi-Diskless, And Diskless Node Configuration

Configuring the disk layout for head and regular nodes is done as part of the initial setup (section 3.3.16 of the *Installation Manual*). For regular nodes, the disk layout can also be re-configured by BCM once the cluster is running. For a head node, however, the disk layout cannot be re-configured after installation by BCM, and head node disk layout reconfiguration must then therefore be treated as a regular Linux system administration task, typically involving backups and resizing partitions.

The remaining parts of this section on disk layouts therefore concern regular nodes, not head nodes.

3.13.1 Disk Layouts

A disk layout is specified using an XML schema (Appendix D.1). The disk layout typically specifies the devices to be used, its partitioning scheme, and mount points. Possible disk layouts include the following:

- Default layout (Appendix D.3)
- Hardware RAID setup (Appendix D.4)
- Software RAID setup (Appendix D.5)
- LVM setup (Appendix D.7)
- Diskless setup (Appendix D.9)
- Semi-diskless setup (Appendix D.10)

3.13.2 Disk Layout Assertions

Disk layouts can be set to *assert*

- that particular hardware be used, using XML element tags such as `vendor` or `requiredSize` (Appendix D.11)
- custom assertions using an XML `assert` element tag to run scripts placed in CDATA sections (Appendix D.12)

3.13.3 Changing Disk Layouts

A disk layout applied to a category of nodes is inherited by default by the nodes in that category. A disk layout that is then applied to an individual node within that category overrides the category setting. This is an example of the standard behavior for categories, as mentioned in section 2.1.3.

By default, the cluster is configured with a standard layout specified in section D.3. The layouts can be accessed from Base View or `cmsh`, and a template layout can be used. A diskless node template can also be used, to change a node from a disked to a diskless node, as illustrated by the example on page 168.

3.13.4 Changing A Disk Layout Using Base View

To change a disk layout with Base View, the current disk layout is accessed by selecting a node category or a specific node from the resource tree in the navigation panel. For a node `node001`, the navigation path is:

`Devices` > `All Devices` > `node001` > `Disk Setup` (a tabbed window)> `Devices` >
and it brings up the window shown in figure 3.12:

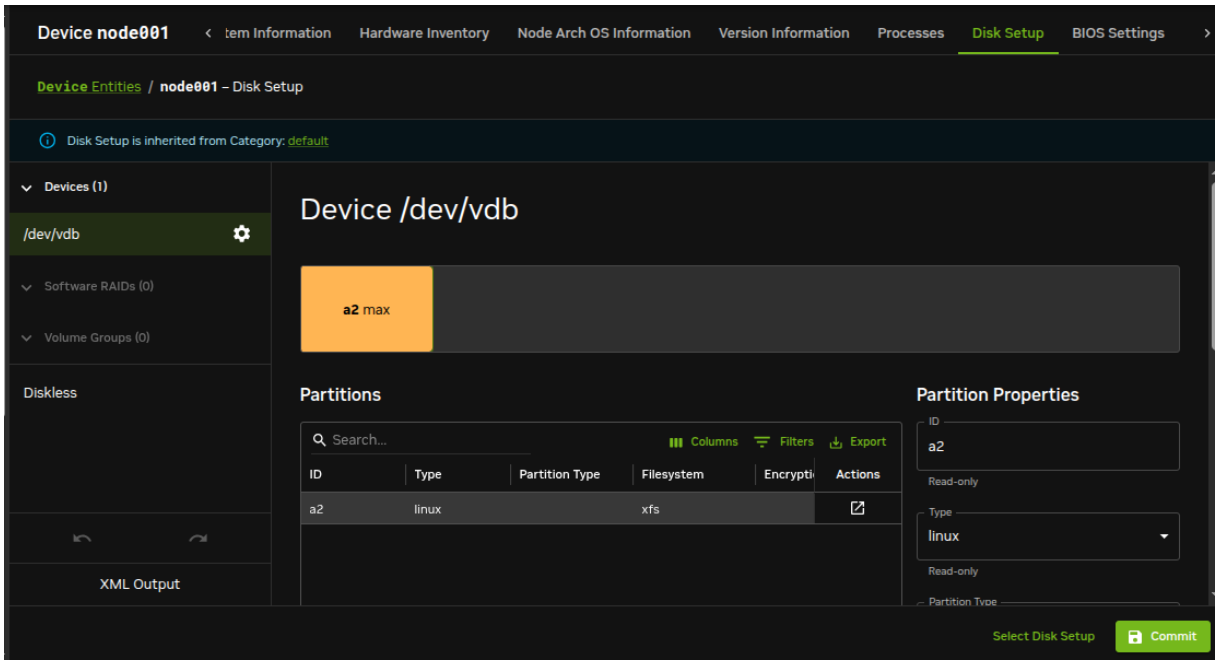


Figure 3.12: Changing a disked node to a diskless node with Base View

However, in a default installation, the values for the devices in the disk layout are inherited from the category level, which can only be modified at category level. That means the default disk layout values are initially read-only values, with the values that were set up during installation (section 3.3.16 of the *Installation Manual*).

At node level, values can be written so that they override the inherited values. Writing values at node level can be done using the `Select Disk Setup` button in various ways:

1. The `Select Disk Setup` button brings up an option to copy from the category.

   If copying over from the category is carried out, then the XML entries in the `Devices` > screen (figure 3.13) can be edited at node level (are no longer read-only):

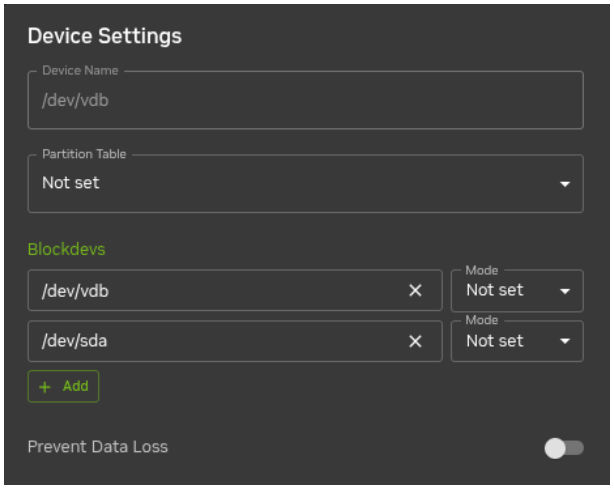


Figure 3.13: Editing disk layouts with Base View

2. The `Select Disk Setup` button also presents a `Select from template` menu option, which shows several possible ready-made configurations that can be loaded up from the CMDaemon database. A selected copy can also be edited to suit the situation.

   The XML schema for a node configured for diskless operation is shown in Appendix D.9. This can often be deployed as is, or it can be modified during deployment using Base View or `cmsh`.

   Changing A Disk Layout From Disked To Diskless

   One of the templates that can be used to switch from the existing disk layout to a diskless one is `slave-diskless.xml`. This XML configuration template can be loaded using the `Select from template` button, and saved to the node or node category.

3. The `Select Disk Setup` button also presents a `Select File` menu option, which can be used to upload a custom configuration via the browser.

3.13.5 Changing A Disk Layout Using `cmsh`

To edit an existing disk layout from within `cmsh`, the existing XML configuration is accessed by editing the `disksetup` property in `device` mode for a particular node, or by editing the `disksetup` property in `category` mode for a particular category. Editing is done using the `set` command, which opens up a text editor:

Example

```
[root@basecm11 ~]# cmsh
[basecm11]% device use node001
[basecm11->device[node001]]% set disksetup
```

After editing and saving the XML configuration, the change is then committed to CMDaemon with the `commit` command. It should be understood that a disk layout XML configuration is not stored in a file on the filesystem, but in the CMDaemon database. The XML configurations that exist on a default cluster at
`/cm/images/default-image/cm/local/apps/cmd/etc/htdocs/disk-setup/`
and
`/cm/local/apps/cmd/etc/htdocs/disk-setup/`
are merely default configurations.

If the `disksetup` setting for a device is deleted, using the `clear` command, then the category level `disksetup` property is used by the device. This is in accordance with the usual behavior for node values

that override category values (section 2.1.5).

Instead of editing an existing disk layout, another XML configuration can also be assigned. A diskless configuration may be chosen and set as follows:

Example

```
[basecm11->device[node001]]% set disksetup /cm/local/apps/cmd/\
etc/htdocs/disk-setup/slave-diskless.xml
```

In these preceding Base View and `cmsh` examples, after committing the change and rebooting the node, the node then functions entirely from its RAM, without using its own disk.

However, RAM is usually a scarce resource, so administrators often wish to optimize diskless nodes by freeing up the RAM on them from the OS that is using the RAM. Freeing up RAM can be accomplished by providing parts of the filesystem on the diskless node via NFS from the head node. That is, mounting the regular node with filesystems exported via NFS from the head node. The details of how to do this are a part of section 3.14, which covers the configuration of NFS exports and mounts in general.

3.14 Configuring NFS Volume Exports And Mounts

NFS allows NFS clients shared access to a filesystem on an NFS server. The accessed filesystem is called an NFS volume by remote machines. The NFS server exports the filesystem to selected hosts or networks, and the clients can then mount the exported volume locally.

An unformatted filesystem cannot be used. The drive must be partitioned beforehand with `fdisk` or similar partitioning tools, and its filesystem formatted with `mkfs` or similar before it can be exported.

In BCM, the head node is typically used to export an NFS volume to the regular nodes, and the regular nodes then mount the volume locally.

- NFS can be made to work at higher speeds with remote direct memory access (RDMA), by bypassing the CPU. If there is RDMA hardware present, and if the `rdma-core` package is installed, then the RDMA service works automatically in RHEL 8 and 9.

  The settings that determine client module loading are set in the file `/etc/rdma/modules/rdma.conf` so that the service auto-loads by default.

- An alternative to NFS over RDMA for very fast file systems is the massively parallel and free (GPLv2) Lustre filesystem, running over InfiniBand.

Automounting With NFS

If auto-mounting is used, then the configuration files for exporting should be set up on the NFS server, and the mount configurations set up on the software images. The service "`autofs`" or the equivalent can be set up using Base View via the "`Services`" option (section 3.15) on the head and regular nodes or node categories. With `cmsh` the procedure to configure auto-mounting on the head and regular nodes could be:

Example

```
[root@basecm11 ~]# cmsh
[basecm11]% device use basecm11
[basecm11->device[basecm11]]% services
[basecm11->device[basecm11]->services]% add autofs
[basecm11->device*[basecm11*]->services*[autofs*]]% show
Parameter                      Value
------------------------------ ------------------------------------
```

```
Autostart                       no
Belongs to role                 no
Monitored                       no
Revision
Run if                          ALWAYS
Service                         autofs
Sickness check interval         60
Sickness check script
Sickness check script timeout   10
Timeout                         -1
[basecm11->device*[basecm11*]->services*[autofs*]]% set autostart yes
[basecm11->device*[basecm11*]->services*[autofs*]]% commit
[basecm11->device[basecm11]->services[autofs]]% category use default
[basecm11->category[default]]% services
[basecm11->category[default]->services]% add autofs
[basecm11->category*[default*]->services*[autofs*]]% set autostart yes
[basecm11->category*[default*]->services*[autofs*]]% commit
[basecm11->category[default]->services[autofs]]%
```

Filesystems imported to a regular node via an auto-mount operation must explicitly be excluded in `excludelistupdate` by the administrator, as explained in section 5.6.1, page 298.

Dropping All Use Of NFS Version 3

At the time of writing, (August 2025), the default NFS protocol versions used in BCM are a mixture of version 3 and 4. This is because some aspects of NFS version 4 clients in BCM have not yet been extensively tested. Migrating the remaining version 3 parts over to version 4 is therefore still experimental. If, despite this, the cluster administrator would still like to migrate over to the use of only NFS version 4 clients, then it can be carried out as follows:

The following configuration files should be modified:

- `/etc/nfs.conf`: On all nodes, changing the default value of `vers3=y` by adding the line

  ```
  vers3=n
  ```

- `/etc/nfsmount.conf`: On the head node(s), changing the line `Nfsvers=3` (the BCM default) to

  ```
  Nfsvers=4
  ```

- `/cm/node-installer*/scripts/node-installer.conf`: On the head node(s), adding the line

  ```
  disableNodeInstallerNFSCertificateStore = true
  ```

- `/cm/node-installer*/scripts/initrd_linuxrc`: On the head node(s), modifying the `nolock` entries:

  ```
  root@basecm11:~# grep nolock /cm/node-installer/scripts/initrd_linuxrc
  # mount with nolock so we do not have to run portmapper+statd
  mount -o nolock,ro $PXE_MASTER_IP:$nodeinstallerpath /installer_root
  mount -vo nolock,ro $PXE_MASTER_IP:$nodeinstallerpath /installer_root
  ```

  so that the `vers=4` option is explicitly specified:

```
root@basecm11:~# grep nolock /cm/node-installer/scripts/initrd_linuxrc
# mount with nolock so we do not have to run portmapper+statd
mount -o nolock,ro,vers=4 $PXE_MASTER_IP:$nodeinstallerpath /installer_root
mount -vo nolock,ro,vers=4 $PXE_MASTER_IP:$nodeinstallerpath /installer_root
```

The changes can be deployed by rebuilding the ramdisk:

Example

```
root@basecm11:~# cmsh
[basecm11]% softwareimage foreach * (createramdisk)
```

and then rebooting the head and regular nodes.

The rest of this section describes the configuration of NFS for static mounts, using Base View or `cmsh`.

Sections 3.14.1 and 3.14.2 explain how exporting and mounting of filesystems is done in general by an administrator using Base View and `cmsh`, and considers some mounting behavior that the administrator should be aware of.

Section 3.14.3 discusses how filesystems in general on a diskless node can be replaced via mounts of NFS exports.

Section 3.14.4 discusses how OFED InfiniBand or iWarp drivers can be used to provide NFS over RDMA.

3.14.1 Exporting A Filesystem Using Base View And `cmsh`

Exporting A Filesystem Using Base View

As an example, if an NFS volume exists at "`basecm11:/modeldata`" it can be exported using Base View using the head node navigation path:

`Devices > All Devices > Head Node >`  `> Settings > Filesystem exports >`

This shows the list of exports (figure 3.14):

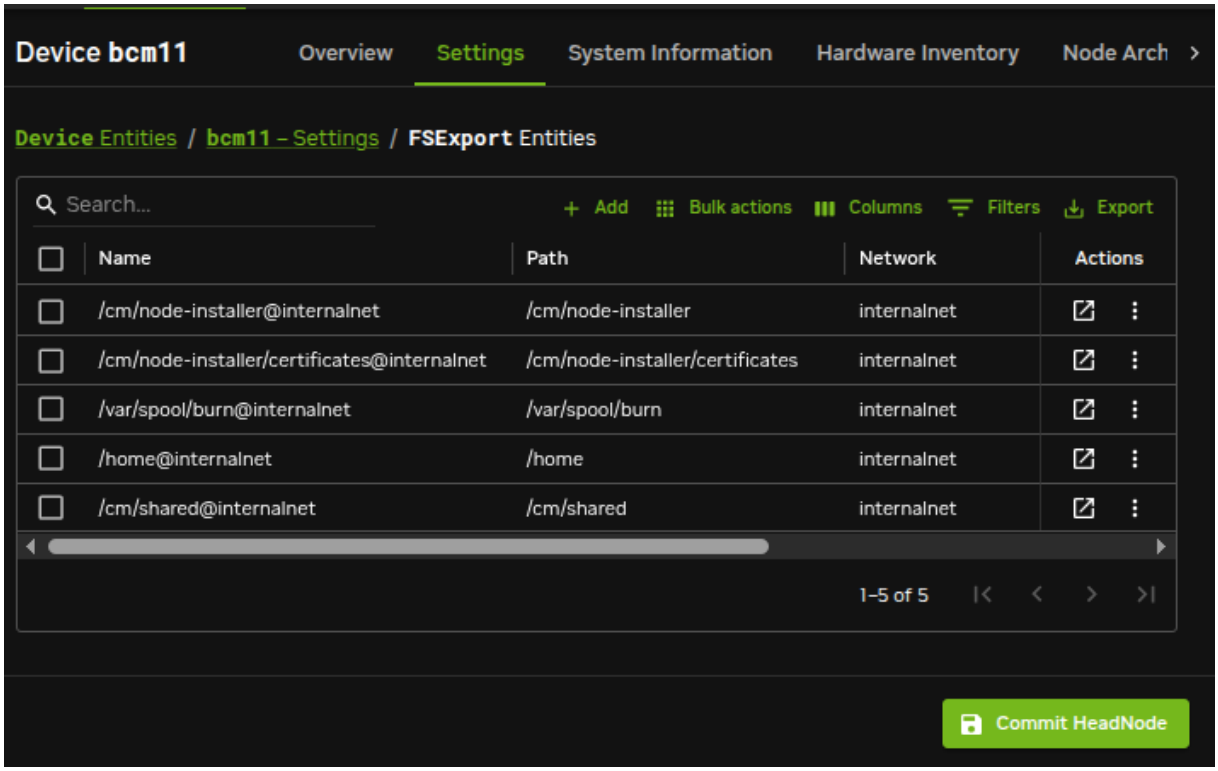


Figure 3.14: NFS exports from a head node viewed using Base View

Using the + Add button, a new entry (figure 3.15) can be configured with values as shown:

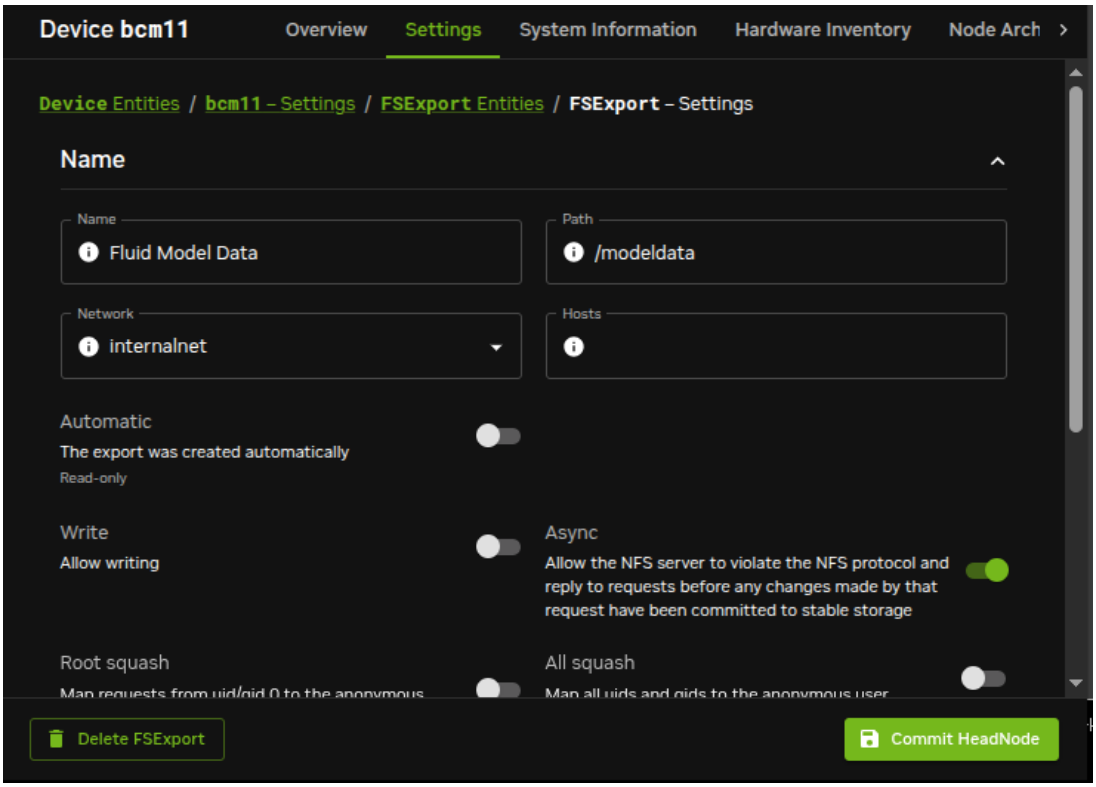


Figure 3.15: Setting up an NFS export using Base View

For this example, the value for "Name" is set arbitrarily to "Fluid Model Data", the value for Path is set to /modeldata, and the value for Network is set from the selection menu to allowing access to internalnet (by default 10.141.0.0/16 in CIDR notation).

By having the Write option disabled, read-only access is kept.

Saving this preceding configuration means the NFS server now provides NFS access to this filesystem for internalnet.

The network can be set to other network values using CIDR notation. It can also be set to particular hosts such as just node001 and node002, by specifying a value of "node001 node002" instead. Other settings and options are also possible and are given in detail in the man pages for exports(5).

Exporting A Filesystem Using cmsh

The equivalent to the preceding Base View NFS export procedure can be done in cmsh by using the fsexports submode on the head node (some output elided):

Example

```
[root@basecm11 ~]# cmsh
[basecm11]% device use basecm11
[basecm11->device[basecm11]]% fsexports
[...->fsexports]% add "Fluid Model Data"
[...->fsexports*[Fluid Model Data*]]% set path /modeldata
[...[Fluid Model Data*]]% set hosts 10.141.0.0/16
[...[Fluid Model Data*]]% commit
[...->fsexports[Fluid Model Data]]% list | grep Fluid
Name (key)           Path            Hosts            Write
-------------------  -------------   ---------------  ------
Fluid Model Data     /modeldata      10.141.0.0/16    no
```

General Considerations On Exporting A Filesystem

Built-in exports: In versions of NVIDIA Base Command Manager prior to version 9.0, all filesystem exports could be removed from the fsexports submode, simply by using the remove command with the name of the export.

From version 9.0 onward however, the following filesystem exports:

- /var/spool/burn
- /home
- /cm/shared

are treated as special built-ins.

Head node role and disableautomaticexports: Built-ins are exported automatically as part of the headnode role, also introduced in NVIDIA Base Command Manager 9.0, and cannot simply be removed.

To disable export of the built-in file systems, the disableautomaticexports command must be run in the headnode role for that node:

Example

```
[root@basecm11 ~]# cmsh
[basecm11]% device use basecm11
[basecm11->device[basecm11]]% roles
[basecm11->device[basecm11]->roles]% use headnode
[basecm11->device[basecm11]->roles[headnode]]% show
Parameter                      Value
------------------------------ ----------------------
```

```
Name                              headnode
Revision
Type                              HeadNodeRole
Add services                      yes
Disable automatic exports         no
role use headnode
[basecm11->device[basecm11]->roles[headnode]]% set disableautomaticexports yes ; commit
```

Disabling exports that are not built-ins: Exports that are not built-ins can still simply be removed. However, also from NVIDIA Base Command Manager 9.0 onward they can also simply be disabled in the `fsexports` submode. For example there is an export created by the cluster administrator for `/opt`, it can be disabled as follows:

Example

```
[basecm11->device[basecm11]->fsexports]% list
Name (key)       Path                Network     Disabled
---------------- -------      ...    ----------- --------
opt              /opt                internalnet no
[basecm11->device[basecm11]->fsexports]% set opt disabled yes; commit
```

The reason for automating the export for nodes via a `headnode` role is that NVIDIA Base Command Manager 9.0 onward has multidistro and multiarch capabilities (section 9.7), which would make manual management of exports harder for such nodes. The reason for the extra hurdle of `disableautomaticexports` for built-ins is that disabling these exports can result in an unbootable system.

3.14.2 Mounting A Filesystem Using Base View And `cmsh`

Continuing on with the `Fluid Model Data` export example from the preceding section, the administrator decides to mount the remote filesystem over the default category of nodes. Nodes can also mount the remote filesystem individually, but that is usually not a common requirement in a cluster. The administrator also decides not to re-use the exported name from the head node. That is, the remote mount name `modeldata` is not used locally, even though NFS allows this and many administrators prefer to do this. Instead, a local mount name of `/modeldatagpu` is used, perhaps because it avoids confusion about which filesystem is local to a person who is logged in, and perhaps to emphasize the volume is being mounted by nodes with GPUs.

Mounting A Filesystem Using Base View

In Base View the navigation path to manage the mount points of nodes in the category of `default` is:

`Grouping > Categories[default] > Settings > Extended settings > Filesystem mounts >`

A mount point can be added with the `+ Add` button. Values for the remote mount point (`basecm11:/modeldata`), the filesystem type (`nfs`), and the local mount point (`/modeldatagpu`) can then be set for the category, while the remaining options stay at their default values (figure 3.16):

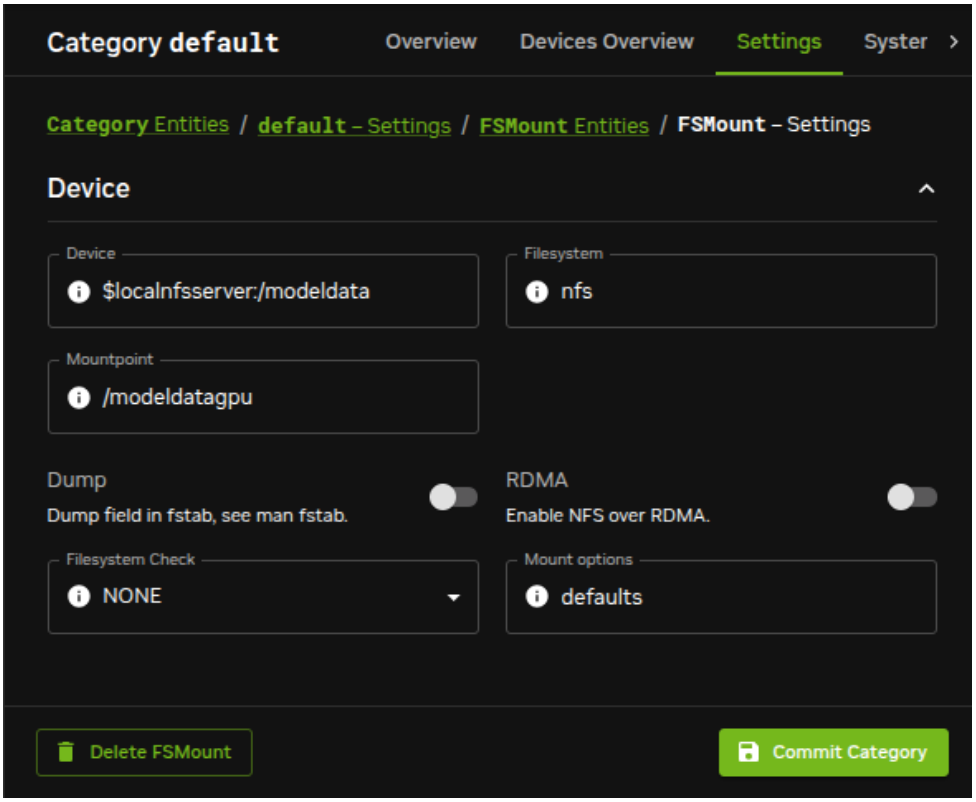


Figure 3.16: Setting up NFS mounts on a node category using Base View

Saving the configuration saves the values and creates the local mount point, so that the volume can then be accessed by nodes within that category.

Mounting A Filesystem Using `cmsh`

The equivalent to the preceding Base View NFS mount procedure can be done in `cmsh` by using the `fsmounts` submode, for example on the `default` category. The `add` method under the `fsmounts` submode sets the mountpoint path, in this case `/modeldatagpu` (some output elided):

Example

```
[root@basecm11 ~]# cmsh
[basecm11]% category use default
[basecm11->category[default]]% fsmounts
[basecm11->category[default]->fsmounts]% add /modeldatagpu
[basecm11->...*[/modeldatagpu*]]% set device basecm11:/modeldata
[basecm11->...*[/modeldatagpu*]]% set filesystem nfs
[basecm11->category*[default*]->fsmounts*[/modeldatagpu*]]% commit
[basecm11->category[default]->fsmounts[/modeldatagpu]]%
Device                  Mountpoint (key)     Filesystem
----------------------  ------------------   ----------
...
basecm11:/modeldata     /modeldatagpu        nfs
[basecm11->category[default]->fsmounts[/modeldatagpu]]% show
Parameter             Value
--------------------  --------------------
Device                basecm11:/modeldata
Dump                  no
Filesystem            nfs
```

```
Filesystem Check      NONE
Mount options         defaults
Mountpoint            /modeldatagpu
```

Values can be set for Mount options other than default. For example, the noac flag can be added as follows:

```
[basecm11->...[/modeldatagpu]]% set mountoptions defaults,noac; commit
```

Mounting a CIFS might use:

```
[basecm11->...[/modeldatagpu]]% set mountoptions gid,users,file_mode=0666,dir_mode=0777,\
iocharset=iso8859-15,credentials=/path/to/credential
[basecm11->...[/modeldatagpu*]]% commit
```

A _netdev mount option to make systemd wait until the network is up before it is mounted can be added as follows:

```
[basecm11->...[/modeldatagpu]]% append mountoptions ,_netdev; commit
```

General Considerations On Mounting A Filesystem

There may be a requirement to segregate the access of nodes. For example, in the case of the preceding, because some nodes have no associated GPUs.

Besides the "Allowed hosts" options of NFS exports mentioned earlier in section 3.14.1, BCM offers two more methods to fine tune node access to mount points:

- Nodes can be placed in another category that does not have the mount point.
- Nodes can have the mount point set, not by category, but per device within the Nodes resource. For this, the administrator must ensure that nodes that should have access have the mount point explicitly set.

Other considerations on mounting are that:

- When adding a mount point object:
  - The settings take effect right away by default on the nodes or node categories.
  - If noauto is set as a mount option, then the option only takes effect on explicitly mounting the filesystem.
  - If "AutomaticMountAll=0" is set as a CMDaemon AdvancedConfig directive (Appendix C, page 932), then CMDaemon changes for /etc/fstab are written to the file, but the mount -a command is not run by CMDaemon. However, the administrator should be aware that since mount -a is run by the distribution during booting, a node reboot implements the mount change.
- While a mount point object may have been removed, umount does not take place until reboot, to prevent mount changes outside of the cluster manager. If a umount needs to be to done without a reboot, then it should be done manually, for example, using the pdsh or pexec command (section 14.1), to allow the administrator to take appropriate action if umounting goes wrong.
- When manipulating mount points, the administrator should be aware which mount points are inherited by category, and which are set for the individual node.
  - In Base View, for a node, inheritance by category is indicated in the navigation path Devices > Nodes[*node name*] > Edit > Settings > Filesystem mounts, under the INHERITED column, with the entry (Category).

– In cmsh, the category a mount belongs to is displayed in brackets. This is displayed from within the fsmounts submode of the device mode for a specified node:

Example

```
[root@basecm11 ~]# cmsh -c "device; fsmounts node001; list"

Device                   Mountpoint (key)     Filesystem
------------------------ -------------------- ----------
[default] none           /dev/pts             devpts
[default] none           /proc                proc
[default] none           /sys                 sysfs
[default] none           /dev/shm             tmpfs
[default] $localnfsserv+ /cm/shared           nfs
[default] basecm11:/home /home                nfs
basecm11:/cm/shared/exa+ /home/examples       nfs
[root@basecm11 ~]#
```

To remove a mount point defined at category level for a node, it must be removed from within the category, and not from the specific node.

Mount Order Considerations

Care is sometimes needed in deciding the order in which mounts are carried out.

- For example, if both /usr/share/doc and a replacement directory subtree /usr/share/doc/compat-gcc-34-3.4.6java are to be used, then the stacking order should be that /usr/share/doc is mounted first. This order ensures that the replacement directory subtree overlays the first mount. If, instead, the replacement directory were the first mount, then it would be overlaid, inaccessible, and inactive.
- There may also be dependencies between the subtrees to consider, some of which may prevent the start up of applications and services until they are resolved. In some cases, resolution may be quite involved.

The order in which such mounts are mounted can be modified with the up and down commands within the fsmounts submode of cmsh.

3.14.3 Mounting A Filesystem Subtree For A Diskless Node Over NFS

NFS Vs tmpfs For Diskless Nodes

For diskless nodes (Appendix D.9), the software image (section 2.1.2) is typically installed from a provisioning node by the node-installer during the provisioning stage, and held as a filesystem in RAM on the diskless node with the tmpfs filesystem type.

It can be worthwhile to replace subtrees under the diskless node filesystem held in RAM with subtrees provided over NFS. This can be particularly worthwhile for less frequently accessed parts of the diskless node filesystem. This is because, although providing the files over NFS is much slower than accessing it from RAM, it has the benefit of freeing up RAM for tasks and jobs that run on diskless nodes, thereby increasing the cluster capacity.

An alternative "semi-diskless" way to free up RAM is to use a local disk on the node itself for supplying the subtrees. This is outlined in Appendix D.10.

Moving A Filesystem Subtree Out Of tmpfs To NFS

To carry out subtree provisioning over NFS, the subtrees are exported and mounted using the methods outlined in the previous examples in sections 3.14.1 and 3.14.2. For the diskless case, the exported filesystem subtree is thus a particular path under /cm/images/*<image>* on the provisioning node, and

the subtree is mounted accordingly under / on the diskless node. By default, <*image*> is `default-image` on a newly-installed cluster.

While there are no restrictions placed on the paths that may be mounted in NVIDIA Base Command Manager 11, the administrator should be aware that mounting certain paths such as `/bin` is not possible.

When Base View or `cmsh` are used to manage the NFS export and mount of the subtree filesystem, then `tmpfs` on the diskless node is reduced in size due to the administrator explicitly excluding the subtree from `tmpfs` during provisioning.

An example might be to export `/cm/images/default-image` from the head node, and mount the directory available under it, `usr/share/doc`, at a mount point `/usr/share/doc` on the diskless node. In `cmsh`, such an export can be done by creating an FS export object corresponding to the software image object `defaultimage` with the following indicated properties (some prompt output elided):

Example

```
[root@basecm11 ~]# cmsh
[basecm11]% device use basecm11; fsexports
[basecm11->device[basecm11]->fsexports]% add defaultimage
[basecm11...defaultimage*]]% set path /cm/images/default-image
[basecm11...defaultimage*]]% set hosts 10.141.0.0/16
[basecm11...defaultimage*]]% commit
[basecm11...defaultimage]]% list | grep defaultimage
Name (key)        Path                      Hosts          Write
---------------   -----------------------   ------------   -----
defaultimage      /cm/images/default-image  10.141.0.0/16  no
```

As the output to `list` shows, the NFS export should be kept read-only, which is the default. Appropriate parts of the export can then be mounted by a node or node category. The mount is defined by setting the mount point, the `nfs` filesystem property, and the export device. For example, for a node category (some output elided):

```
[br...defaultimage]]% category use default
[basecm11->category[default]]% fsmounts
[basecm11->category[default]->fsmounts]% add /usr/share/doc
[basecm11->...*[/usr/share/doc*]]% set device basecm11:/cm/images/default-image/usr/share/doc
[basecm11->...*[/usr/share/doc*]]% set filesystem nfs
[basecm11->category*[default*]->fsmounts*[/usr/share/doc*]]% commit
[basecm11->category[default]->fsmounts[/usr/share/doc]]% list
Device                                          Mountpoint (key)     Filesystem
----------------------                          ------------------   ----------
...                                             ...                  ...
basecm11:/cm/images/default-image/usr/share/doc /usr/share/doc       nfs
[basecm11->category[default]->fsmounts[/usr/share/doc]]% show
Parameter          Value
---------------    ----------------------------------------------
Device             basecm11:/cm/images/default-image/usr/share/doc
Dump               no
Filesystem         nfs
Filesystem Check   0
Mount options      defaults
Mountpoint         /usr/share/doc
```

Other mount points can be also be added according to the judgment of the system administrator. Some consideration of mount order may be needed, as discussed on page 177 under the subheading "Mount Order Considerations".

An Example Of Several NFS Subtree Mounts

The following mounts save about 440MB from `tmpfs` on a diskless node with Rocky Linux 8, as can be worked out from the following subtree sizes:

```
[root@basecm11 ~]# cd /cm/images/default-image/
[root@basecm11 default-image]# du -sh usr/share/locale usr/lib/jvm usr/share/doc usr/src
160M    usr/share/locale
118M    usr/lib/jvm
88M    usr/share/doc
77M     usr/src
```

The filesystem mounts can then be created using the techniques in this section. After doing that, the result is then something like (some lines omitted):

```
[root@basecm11 default-image]# cmsh
[basecm11]% category use default; fsmounts
[basecm11->category[default]->fsmounts]% list -f device:53,mountpoint:17
device                                               mountpoint (key)
-------------------------------------------------- -----------------
...                                                ...
master:/cm/shared                                  /cm/shared
master:/home                                       /home
basecm11:/cm/images/default-image/usr/share/locale /usr/share/locale
basecm11:/cm/images/default-image/usr/lib/jvm      /usr/lib/jvm
basecm11:/cm/images/default-image/usr/share/doc    /usr/share/doc
basecm11:/cm/images/default-image/usr/src          /usr/src
[basecm11->category[default]->fsmounts]%
```

Diskless nodes that have NFS subtree configuration carried out on them can be rebooted to start them up with the new configuration.

3.14.4 Configuring NFS Volume Exports And Mounts Over RDMA With OFED Drivers

If running NFS over RDMA, then at least NFS version 4.0 is recommended. NFS version 3 will also work with RDMA, but uses IPoIB encapsulation instead of native verbs. NFS version 4.1 uses the RDMA Connection Manager (`librdmacm`), instead of the InfiniBand Connection Manager (`ib_cm`) and is thereby also able to provide pNFS.

The administrator can set the version of NFS used by the cluster by setting the value of `Nfsvers` in the file `/etc/nfsmount.conf` on the head node(s).

Drivers To Use For NFS Over RDMA

The DOCA Mellanox OFED drivers (Chapter 10 of the *Installation Manual*) can support using the RDMA protocol (section 3.6) to provide NFS.

The distribution OFED drivers also support NFS over RDMA.

When using NFS over RDMA, `ibnet`, the IP network used for InfiniBand, should be set. Section 3.6.4 explains how that can be done.

Exporting With Base View And `cmsh` Using NFS Over RDMA

With the drivers installed, a volume export can be carried out using NFS over RDMA.

The procedure using Base View is much the same as done in section 3.14.1 ("Exporting A Filesystem Using Base View"), except for that the `ibnet` network should be selected instead of the default `internalnet`, and the "`RDMA`" option should be enabled.

The procedure using `cmsh` is much the same as done in section 3.14.1 ("Exporting A Filesystem Using `cmsh`"), except that the `ibnet` network (normally with a recommended value of 10.149.0.0/16) should be set, and the `rdma` option should be set.

Example

(based on the example in section 3.14.1)

```
...
[...->fsexports*[Fluid Model Data*]]% set path /modeldata
[...[Fluid Model Data*]]% set hosts ibnet
[...[Fluid Model Data*]]% set rdma yes
[...[Fluid Model Data*]]% commit
...
```

Mounting With Base View And `cmsh` Using NFS Over RDMA

The mounting of the exported filesystems using NFS over RDMA can then be done.

The procedure using Base View is largely like that in section 3.14.2, ("Mounting A Filesystem Using Base View"), except that the `Device` entry must point to `master.ib.cluster` so that it resolves to the correct NFS server address for RDMA, and the checkbox for `NFS over RDMA` must be ticked.

The procedure using `cmsh` is similar to that in section 3.14.2, ("Mounting A Filesystem Using `cmsh`"), except that device must be mounted to the `ibnet`, and the `rdma` option must be set, as shown:

Example

(based on the example in section 3.14.2)

```
...
[basecm11->category[default]->fsmounts]% add /modeldatagpu
[basecm11->...*[/modeldatagpu*]]% set device basecm11.ib.cluster:/modeldata
[basecm11->...*[/modeldatagpu*]]% set filesystem nfs
[basecm11->...*[/modeldatagpu*]]% set rdma yes
[basecm11->category*[default*]->fsmounts*[/modeldatagpu*]]% commit
...
```

3.15 Managing And Configuring Services

3.15.1 Why Use The Cluster Manager For Services?

Linux administrators should be familiar with managing services from the command line using `systemctl`:

Example

```
systemctl start <service~name>.service
```

where <*service name*> indicates a service such as `mysqld`, `mariabd`, `nfs`, `postfix` and so on.

Services can also be managed with BCM. That is, they can also be started and stopped with Base View and `cmsh` tools.

An additional convenience that comes with the cluster manager tools is that some CMDaemon parameters useful for managing services in a cluster are very easily configured, whether on the head node, a regular node, or for a node category. These parameters are:

- `monitored`: checks periodically if a service is running. Information is displayed and logged the first time it starts or the first time it dies
- `autostart`: restarts a failed service that is being `monitored`.
  - If `autostart` is set to `on`, and a service is stopped using BCM, then no attempts are made to restart the service. Attempted autostarts become possible again only after BCM starts the service again.
  - If `autostart` is set to `on`, and if a service is removed using BCM, then the service is stopped before removal.

  - If `autostart` is `off`, then a service that has not been stopped by CMDaemon still undergoes an attempt to restart it, if
    * CMDaemon is restarted
    * its configuration files are updated by CMDaemon, for example in other modes, as in the example on page 110.

- `runif`: (only honored for nodes used as part of a high availability configuration (Chapter 15)) whether the service should run with a state of:
  - `active`: run on the active node only
  - `passive`: run on the passive only
  - `always`: run both on the active and passive
  - `preferpassive`: preferentially run on the passive if it is available. Valid only for head nodes. Invalid for failover groups (section 15.5.3).

  BCM can be used to keep a service working across a failover event with an appropriate `runif` setting and appropriate failover scripts such as the `Prefailover script` and the `Postfailover script` (section 15.4.6). The details of how to do this will depend on the service.

The details of a service configuration remain part of the configuration methods of the service software itself.

- Thus BCM can run actions on typical services only at the generic service level to which all the unix services conform. This means that CMDaemon can run actions such as starting and stopping the service. If the restarting action is available in the script, then CMDaemon can also run that.
- The operating system configuration of the service itself, including its persistence on reboot, remains under control of the operating system, and is not handled by CMDaemon. So, stopping a service within CMDaemon means that by default the service may start up on reboot. Running

  `systemctl disable <`*service name*`>.service`

  from the command line can be used to configure the service to no longer start up on reboot.

For convenience, BCM provides a `systemctl` command in `cmsh` to provide an overview of the status of units such as services (section 3.15.2, page 185).

3.15.2 Managing And Configuring Services—Examples

If, for example, the CUPS software is installed in an image with the help of `cm-chroot-sw-img`, and the nodes rebooted, then the CUPS service can be managed in several ways:

Managing The Service From The Regular Shell, Outside Of CMDaemon

Standard unix commands from the bash prompt work, as shown by this session:

```
[root@basecm11 ~]# systemctl enable cups.service
... symlinks created...
[root@basecm11 ~]# systemctl start cups
```

Managing The Service From `cmsh`

Listing existing services managed by `cmsh`: Services that are managed by `cmsh` can be listed by the `list` command in `services` mode at `device` level. They mostly run on the head node. For example, on a standard BCM head node without workload managers or Kubernetes:

Example

```
[basecm11->device[basecm11]->services]% list
Service (key)                       Monitored  Autostart
----------------------------------- ---------- ----------
[default] cuda-dcgm                 yes        yes
[default] mst                       yes        yes
[default] nslcd                     yes        yes
[default] rshim                     yes        yes
[role: boot, role: headnode] ntpd   yes        yes
[role: boot] dhcpd                  yes        yes
[role: headnode] conman             yes        yes
[role: headnode] named              yes        yes
[role: headnode] postfix            yes        yes
[role: headnode] shorewall          yes        yes
[role: headnode] shorewall6         yes        yes
[role: headnode] slapd              yes        yes
[role: storage] nfs                 yes        yes
```

These services are typically provided by BCM roles (section 2.1.5). For example, assigning the `boot` role to a node means that it runs the `dhcpd` service, which allows the netbooting of nodes from that node.

The services not managed by BCM are not listed in `services` mode, until they are configured to be managed by BCM. For example, `cron` is not listed in `services` mode unless it is added to `services` mode.

The `overview` command displays the `list` command output in a different way, along with the `runif` property.

Example

```
[basecm11->device[basecm11]->services]% overview
Name         From                        Monitored    Autostart    Run if
------------ --------------------------- ------------ ------------ --------
conman       role: headnode              yes          yes          active
cuda-dcgm    default                     yes          yes          always
dhcpd        role: boot                  yes          yes          always
...
```

Adding and starting the service in `cmsh`: The following session illustrates adding the CUPS service from within `device` mode and the `services` submode. The device in this case is a regular node, `node001`, but a head node can also be chosen. Monitoring and auto-starting are also set in the session:

```
[basecm11]% device services node001
[basecm11->device[node001]->services]% add cups
[basecm11->device*[node001*]->services*[cups*]]% show
Parameter                      Value
------------------------------ ------------------------------------------------
Revision
Service                        cups
Run if                         always
Monitored                      no
Autostart                      no
Managed                        no
Sickness check script
Sickness check script timeout  10
Sickness check interval        60
Timeout                        -1
[basecm11->device*[node001*]->services*[cups*]]% set monitored on
```

```
[basecm11->device*[node001*]->services*[cups*]]% set autostart on
[basecm11->device*[node001*]->services*[cups*]]% commit
[basecm11->device[node001]->services[cups]]%
Tue Sep 30 14:28:26 2025 [notice] node001: Service cups was started
[basecm11->device[node001]->services[cups]]%
```

Other service options in `cmsh`: Within the `services` submode of `cmsh`, the commands:

- `start`
- `stop`
- `restart`
- `update`
- `reload`

can be used to manage the service.

These commands correspond to the modern `systemctl` options, as in the syntax:

`systemctl start|stop|restart|...` <*service name*>

For example, continuing with the preceding session, stopping the CUPS service can be done by running the `stop` option as follows:

Example

```
[basecm11->device[node001]->services[cups]]% stop
Wed Oct  1 12:30:36 2025 [notice] node001: Service cups was stopped
Successfully stopped service cups on: node001
```

The service is then in a `STOPPED` state according to the `status` command.

```
[basecm11->device[node001]->services[cups]]% status
cups                [STOPPED]
```

Details on how a state is used when monitoring a service are given in the section *Monitoring A Service With* `cmsh` *And Base View* on page 187.

Continuing from the preceding session, the CUPS service can also be added for a node category from `category` mode:

```
[basecm11->device[node001]->services[cups]]% category
[basecm11->category]% services default
[basecm11->category[default]->services]% add cups
```

As before, after adding the service, the monitoring and autostart parameters can be set for the service. Also as before, the options to the `service` <*service name*> `start|stop|restart|...` command can be used to manage the service at the `services` submode level. The settings apply to the entire node category (some lines elided):

Example

```
[basecm11->category*[default*]->services*[cups*]]% show
Parameter                      Value
------------------------------ ------------------------------------------------
Revision
Service                        cups
Run if                         always
Monitored                      no
```

```
Autostart                        no
Managed                          no
Sickness check script
Sickness check script timeout    10
Sickness check interval          60
Timeout                          -1
[basecm11->category*[default*]->services*[cups*]]% set autostart yes
[basecm11->category*[default*]->services*[cups*]]% set monitored yes
[basecm11->category*[default*]->services*[cups*]]% commit
[basecm11->category[default]->services[cups]]%
Fri Apr 14 14:06:27 2017 [notice] node002: Service cups was started
Fri Apr 14 14:06:27 2017 [notice] node005: Service cups was started
Fri Apr 14 14:06:27 2017 [notice] node004: Service cups was started
Fri Apr 14 14:06:27 2017 [notice] node003: Service cups was started
[basecm11->category[default]->services[cups]]% list
Service (key)            Monitored  Autostart
------------------------ ---------- ----------
cups                     yes        yes
[basecm11->category[default]->services[cups]]% status
node001..................... cups            [STOPPED ]
node002..................... cups            [   UP   ]
node003..................... cups            [   UP   ]
node004..................... cups            [   UP   ]
node005..................... cups            [   UP   ]
```

Unlike at `device` level (page 181), the `list` command for the `services` submode at `category` level lists only the services that have been manually-configured by `cmsh` or Base View in that mode. So, services defined by roles are not included in the category-level list.

The `status` command lists all the services managed by BCM in a category, which includes the ones defined by roles:

Example

```
[basecm11 ->category[default]->services]% list
Service (key)            Monitored  Autostart
------------------------ ---------- ----------
cups                     yes        yes
[basecm11->category[default]->services]% status
Node         Service      Status
------------ ------------ -----------
node001      cuda-dcgm    [NOT USED]
node001      cups         [   UP   ]
node001      mst          [NOT USED]
node001      nslcd        [   UP   ]
node001      rshim        [NOT USED]
node002      cuda-dcgm    [NOT USED]
node002      cups         [FAILING ]
node002      mst          [NOT USED]
...
```

Removing A Service From CMDaemon Control Without Shutting It Down

Removing a service from BCM control while `autostart` is set to `on` stops the service on the nodes:

```
[basecm11->category[default]->services]% add cups
[basecm11->category*[default*]->services*[cups*]]% set monitored on
[basecm11->category*[default*]->services*[cups*]]% set autostart on
```

```
[basecm11->category*[default*]->services*[cups*]]% commit; exit
[basecm11->category[default]->services]% remove cups; commit
Wed May 23 12:53:58 2012 [notice] node001: Service cups was stopped
```

In the preceding cmsh session, cups starts up when the autostart parameter is committed, if cups is not already up.

The behavior of having the service stop on removal is implemented because it is usually what is wanted.

However, sometimes the administrator would like to remove the service from CMDaemon control without it shutting down. To do this, autostart must be set to off first.

```
[basecm11->category[default]->services]% add cups
[basecm11->category*[default*]->services*[cups*]]% set monitored on
[basecm11->category*[default*]->services*[cups*]]% set autostart off
[basecm11->category*[default*]->services*[cups*]]% commit; exit
Wed May 23 12:54:40 2012 [notice] node001: Service cups was started
[basecm11->category[default]->services]% remove cups; commit
[basecm11->category[default]->services]% !# no change: cups stays up
```

Querying Service Information With systemctl

Services are the most common type of systemd unit. The systemctl command displays the status of systemd units on a device.

A service status overview can be carried out on devices using the BCM systemctl command in cmsh device mode. Running it presents information about services, mounts, slices, timers, and targets including their load state, active state, and sub-state:

Example

```
[basecm11->device[node001]]% systemctl
Node     Unit                       Load     Active   Sub        Description
-------- ------------------------ -------- -------- ---------- ---------------------------------------
node001  -.mount                  loaded   active   mounted    Root Mount
node001  -.slice                  loaded   active   active     Root Slice
node001  ModemManager.service     loaded   active   running    Modem Manager
node001  apache2.service          loaded   active   running    The Apache HTTP Server
node001  apt-daily-upgrade.timer  loaded   active   waiting    Daily apt upgrade and clean activities
node001  apt-daily.timer          loaded   active   waiting    Daily apt download activities
node001  basic.target             loaded   active   active     Basic System
node001  binfmt-support.service   loaded   active   exited     Enable support for additional executabl+
node001  blk-availability.service loaded   active   exited     Availability of block devices
node001  bmc_configure.service    loaded   active   exited     Configure BMC
node001  boot-efi.mount           loaded   active   mounted    /boot/efi
node001  boot.mount               loaded   active   mounted    /boot
node001  cm-nfs-checker.service   loaded   active   running    BCM NFS monitor and checker
node001  cmd.service              loaded   active   running    BCM daemon
...
```

The systemctl command has options to run it per node, category, rack, and so on. Details on the options can be seen by running the help command (help systemctl).

Managing The Service From Base View

Using Base View, a service can be managed from an OSServiceConfig list window, accessible via the Services section of the Settings tab window. The OSServiceConfig list window is accessible for

- Head nodes, for example using a navigation path of
  Devices > All Devices > Head Node > Settings > Services

- regular nodes, for example for `node001` using a navigation path of `Devices > All Devices > node001` > `Settings > Services` (figure 3.17):

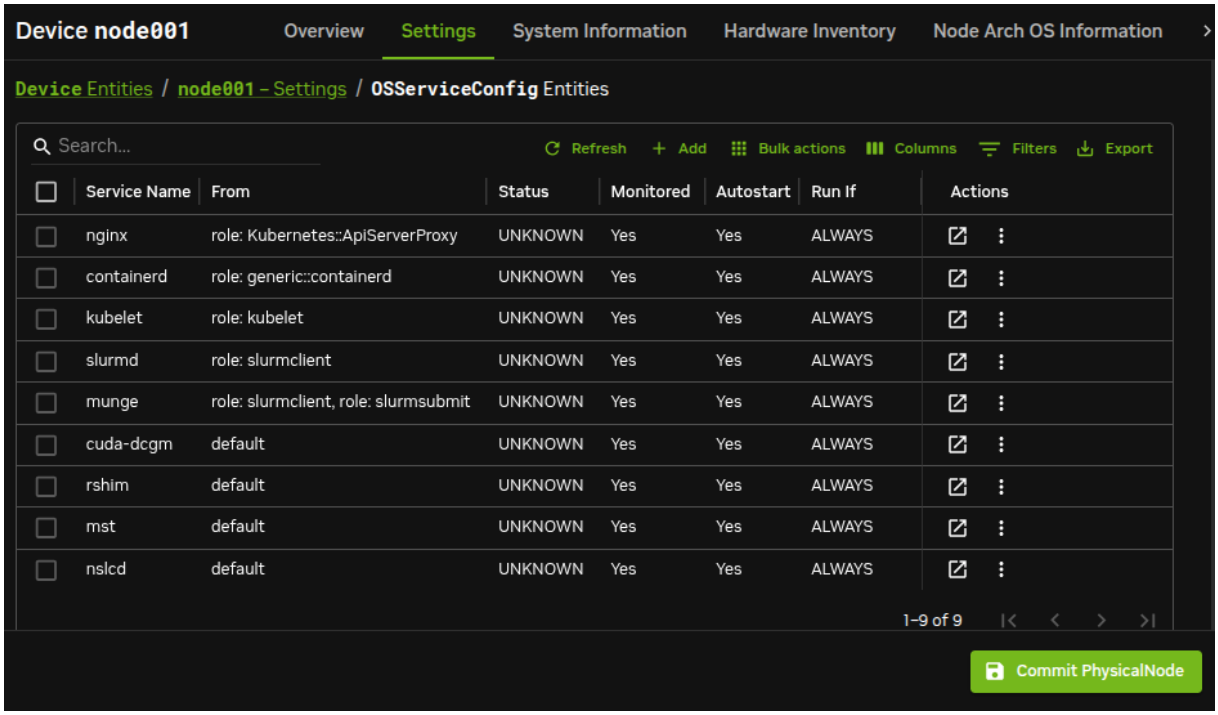


Figure 3.17: Operating system service configuration list window for nodes in Base View

- Node categories, for example using a navigation path of `Grouping > Categories` > `Settings > Services`

A service can be added with the `+Add` button, and fields of the service can then be edited. Figure 3.18 shows CUPS in the services window of Base View (figure 3.18).

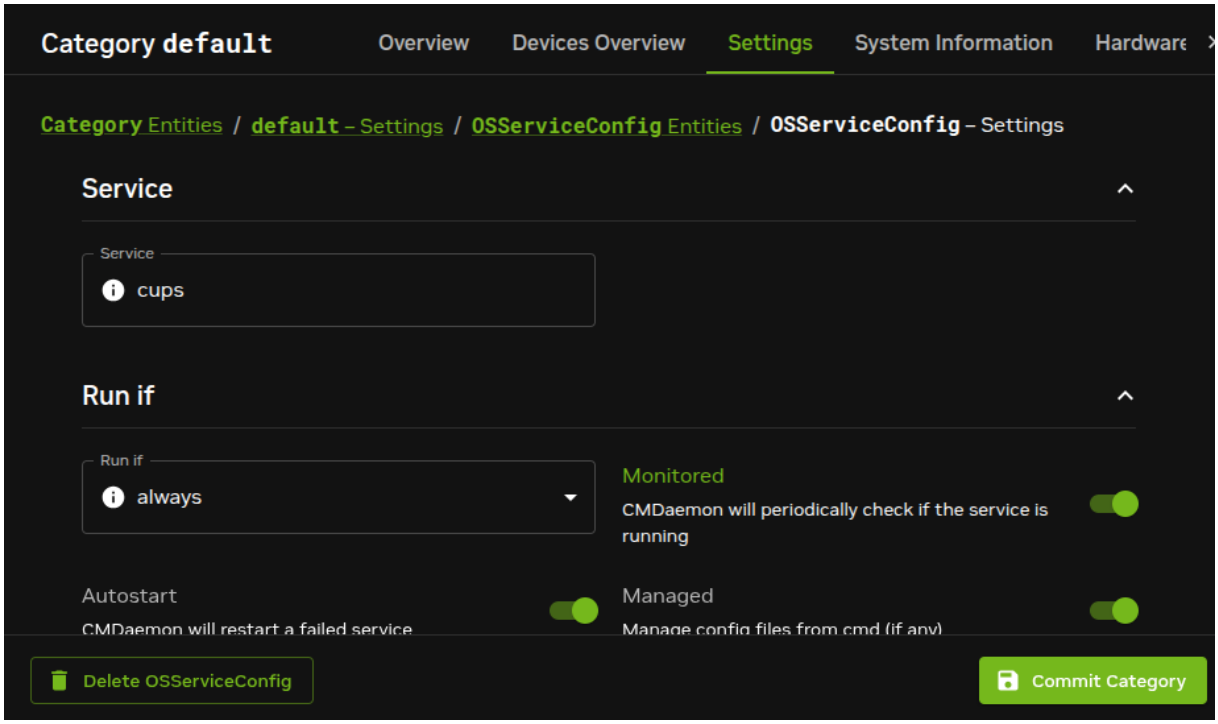


Figure 3.18: Setting up a service using Base View

The service command options `start`, `stop`, `restart`, and so on, are displayed as options when clicking on the ⋮ in the `Actions` column (figure 3.19):

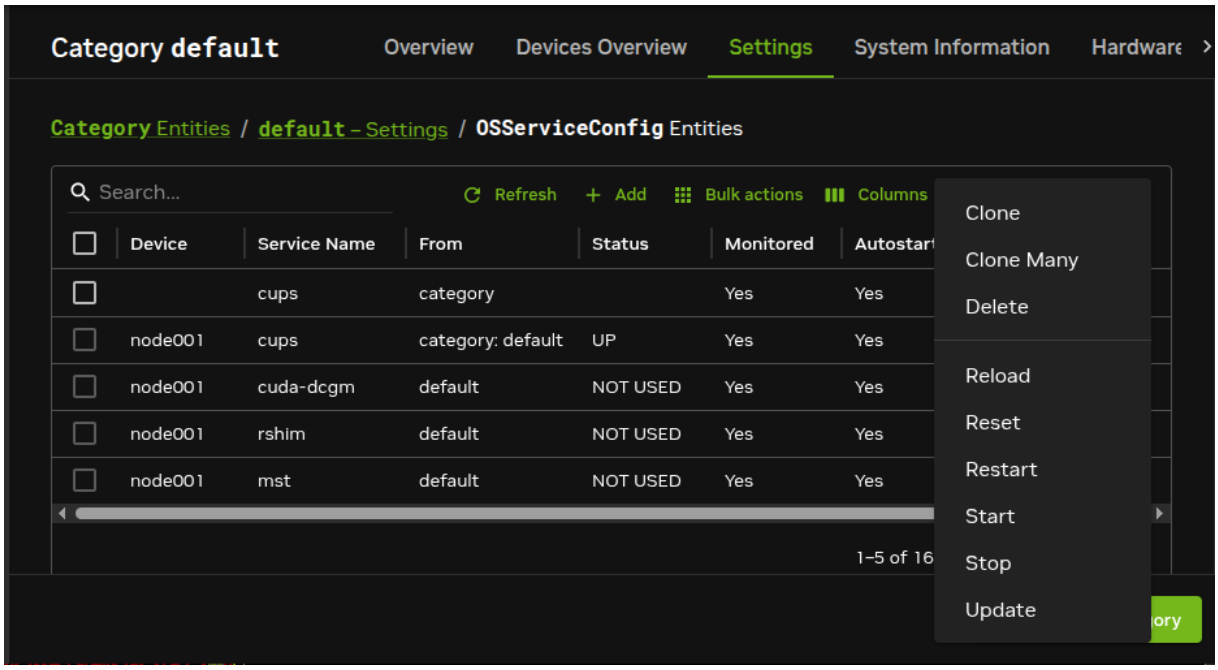


Figure 3.19: Operating system service options, seen after clicking on ⋮ in Actions column in Base View

Monitoring A Service With `cmsh` And Base View

The service is

- in a DOWN state if it is not running, and
- in a FAILING state if it is unable to run after 10 auto-starts in a row

Event messages are sent during these first 10 auto-starts. After the first 10 auto-starts, no more event messages are sent, but autostart attempts continue.

In case an autostart attempt has not yet restarted the service, the reset option may be used to attempt an immediate restart. The reset option is not a service option in the regular shell, but is used by CMDaemon (within cmsh and Base View) to clear a FAILING state of a service, reset the attempted auto-starts count to zero, and attempt a restart of the service.

The monitoring system sets the ManagedServicesOk health check (Appendix G.2.1) to a state of FAIL if any of the services it monitors is in the FAILING state. In cmsh, the statuses of the services are listed by running the latesthealthdata command (section 10.6.3) from device mode.

Standard init.d script behavior is that the script return a non-zero exit code if the service is down, and a zero exit code if the service is up. A non-zero exit code makes BCM decide that the service is down, and should be restarted.

However, some scripts return a non-zero exit code even if the service is up. These services therefore have BCM attempt to start them repetitively, even though they are actually running.

This behavior is normally best fixed by setting a zero exit code for when the service is up, and a non-zero exit code for when the service is down.

3.16 Managing And Configuring A Rack

3.16.1 Racks

A cluster may have local nodes grouped physically into racks. A rack is 42 units in height by default, and nodes normally take up one unit.

Rack List

Rack list in Base View: The Rack list pane can be opened up in Base View via the navigation path Datacenter Infrastructure > Racks (figure 3.20):

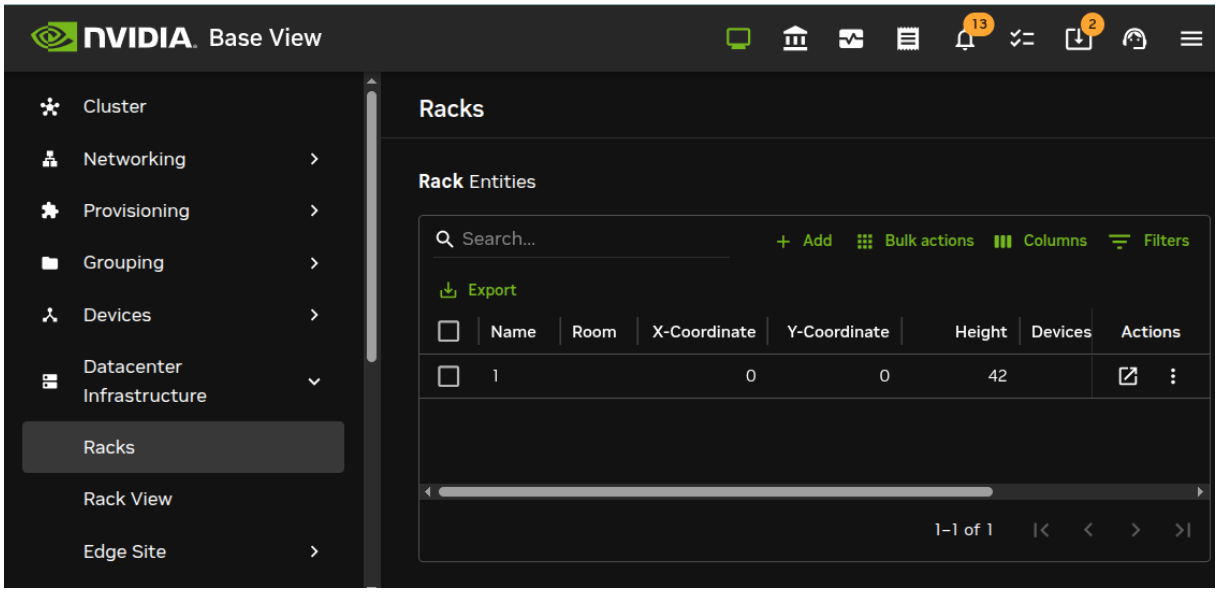


Figure 3.20: Rack list using Base View

Racks can then be added, removed, or edited from the pane.

Within the Rack list pane:

- a new rack item can be added with the + Add button. This opens up a rack configuration window where details of the new configuration can be entered and saved (figure 3.21).
- an existing rack item can be edited with the navigation path > Settings. This opens up the Settings tab from within which rack configuration details can be managed.

Racks overviews in cmsh:

- The list command in rack mode in cmsh allows racks defined in the cluster manager to be listed:

```
[basecm11->rack]% list
Name (key)      Room           x-Coordinate   y-Coordinate   Height
--------------  -------------  -------------  -------------  ------
rack2           skonk works    2              0              42
racknroll                      1              0              42
```

- The rackoverview command displays information about the types of entities in a specified rack. It also lists some more detailed information about some of the entity types:

Example

```
[basecm11->rack]% rackoverview <TAB><TAB>
a01  a02  a03  a04  a05  a06  a07  a08  a09  a10  a11  a12  b01  b02  b03  b04...
[basecm11->rack]% rackoverview a05
Type              Up              Down            Closed          Total
----------------- --------------- --------------- --------------- ---------------
Nodes             18              0               0               18
DPU nodes         0               0               0               0
Managed switches  0               0               0               0
NVLink switches   0               9               0               9
Power shelves     8               0               0               8
Devices           0               0               0               0
Cores             2,592           -               -               2,592
GPUs              72              -               -               72

Name                         Value
---------------------------- -----------------
User CPU                     0.03%
System CPU                   0.07%
Idle CPU                     99.9%
Other CPU                    0.0%
Memory used                  1.28 TiB (4.53%)
Memory unused                27.4 TiB (96.6%)
Memory total                 28.3 TiB
Total GPU utilization        0 W
Total GPU power usage        10.9 KW
Total GPU NVlink bandwidth   0 W
Average GPU temperature      30.5588 C

Node               GPU  Utilization  Temperature  Power usage  Memory used  Memory free  Fabric status
------------------ ---- ------------ ------------ ------------ ------------ ------------ -------------
a05-p1-dgx-01-c01  gpu0 0.0%         30 C         166.551 W    0 B          185 GiB      success
a05-p1-dgx-01-c01  gpu1 0.0%         30 C         158.166 W    1.00 MiB     185 GiB      success
...
```

```
Switch          Utilization  Temperature  Power usage  Fan speed    Links active Links inactive
--------------- ------------ ------------ ------------ ------------ ------------ --------------
a05-p1-nvsw-01  35.7%        0 C          0 W          0 RPM        0            0
a05-p1-nvsw-02  26.9%        0 C          0 W          0 RPM        0            0
...

Power           Input    Output                Fan       Active  Total
shelf           power    power    Temperature  speed     PSU     PSU
--------------- -------- --------  ----------  --------- ------- ------
a05-p1-pwr-01   0 W      0 W      0 C          0 RPM     0       6
a05-p1-pwr-02   0 W      0 W      0 C          0 RPM     0       6
...
[basecm11->rack]%
```

- The `display` command in `rack` mode is useful for visualizing where devices are located in the rack, if the cluster administrator has recorded the device positions

Example

```
[basecm11->rack]% display |less -R
...
            A05                     B05

48                      48
47                      47
46                      46
45                      45
44                      44
43                      43
42   a05-p1-pwr-08      42   B05-P1-PWR-08
41   a05-p1-pwr-07      41   B05-P1-PWR-07
40   a05-p1-pwr-06      40   B05-P1-PWR-06
39   a05-p1-pwr-05      39   B05-P1-PWR-05
38                      38
37   a05-p1-dgx-01-c18  37   b05-p1-dgx-05-c18
36   a05-p1-dgx-01-c17  36   b05-p1-dgx-05-c17
35   a05-p1-dgx-01-c16  35   b05-p1-dgx-05-c16
34   a05-p1-dgx-01-c15  34   b05-p1-dgx-05-c15
33   a05-p1-dgx-01-c14  33   b05-p1-dgx-05-c14
32   a05-p1-dgx-01-c13  32   b05-p1-dgx-05-c13
31   a05-p1-dgx-01-c12  31   b05-p1-dgx-05-c12
30   a05-p1-dgx-01-c11  30   b05-p1-dgx-05-c11
29   a05-p1-dgx-01-c10  29   b05-p1-dgx-05-c10
28   a05-p1-dgx-01-c09  28   b05-p1-dgx-05-c09
27   a05-p1-nvsw-09     27   B05-P1-NVSW-09
26   a05-p1-nvsw-08     26   B05-P1-NVSW-08
25   a05-p1-nvsw-07     25   B05-P1-NVSW-07
24   a05-p1-nvsw-06     24   B05-P1-NVSW-06
23   a05-p1-nvsw-05     23   B05-P1-NVSW-05
22   a05-p1-nvsw-04     22   B05-P1-NVSW-04
21   a05-p1-nvsw-03     21   B05-P1-NVSW-03
20   a05-p1-nvsw-02     20   B05-P1-NVSW-02
19   a05-p1-nvsw-01     19   B05-P1-NVSW-01
18   a05-p1-dgx-01-c08  18   b05-p1-dgx-05-c08
17   a05-p1-dgx-01-c07  17   b05-p1-dgx-05-c07
```

```
16   a05-p1-dgx-01-c06   16   b05-p1-dgx-05-c06
15   a05-p1-dgx-01-c05   15   b05-p1-dgx-05-c05
14   a05-p1-dgx-01-c04   14   b05-p1-dgx-05-c04
13   a05-p1-dgx-01-c03   13   b05-p1-dgx-05-c03
12   a05-p1-dgx-01-c02   12   b05-p1-dgx-05-c02
11   a05-p1-dgx-01-c01   11   b05-p1-dgx-05-c01
10                       10
09   a05-p1-pwr-04       09   B05-P1-PWR-04
08   a05-p1-pwr-03       08   B05-P1-PWR-03
07   a05-p1-pwr-02       07   B05-P1-PWR-02
06   a05-p1-pwr-01       06   B05-P1-PWR-01
05                       05
04                       04
03                       03
02                       02
01                       01
```

Other rack overview commands allow the electrical supply, liquid cooling, leak detection, and leak detection-related actions to be viewed at rack level (section 3.3 of the *NVIDIA Mission Control Manual*).

Rack Configuration Settings

Rack configuration settings in Base View: A `Settings` tab for editing a rack item selected from the `Rack list` pane is shown in figure 3.21.

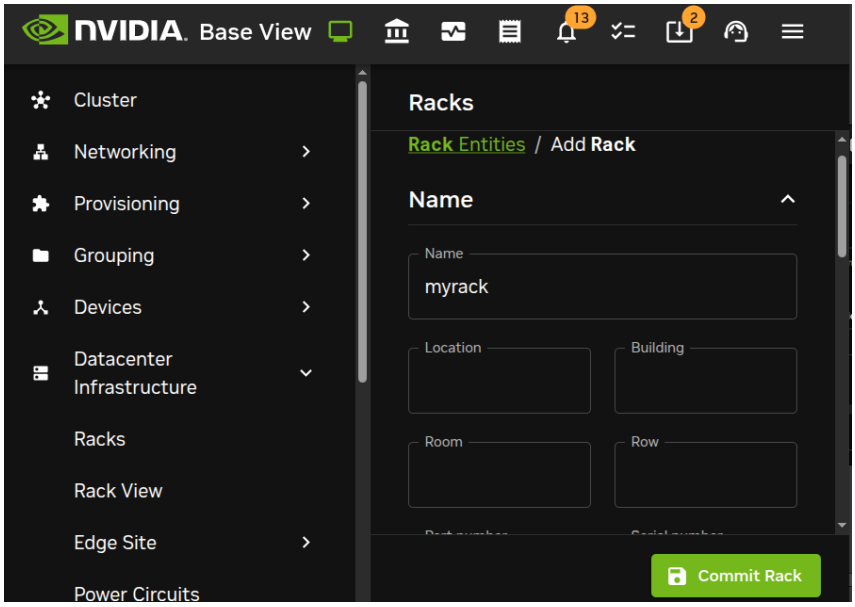


Figure 3.21: Rack configuration settings using Base View

Among the rack configuration attributes are:

- `Name`: A unique name for the rack item. Names such as `rack001`, `rack002` are a sensible choice.
- `Room`: A unique name for the room the rack is in.
- `Position`: The x- and y-coordinates of the rack in a room. These coordinates are meant to be a hint for the administrator about the positioning of the racks in the room, and as such are optional, and can be arbitrary numbers. The `Notes` field can be used as a supplement or as an alternative for hints.

- `Height`: by default this is the standard rack size of 42U.
- `Inverted`: Normally, a rack uses the number 1 to mark the top and 42 to mark the bottom position for the places that a device can be positioned in a rack. However, some manufacturers invert this and use 1 to mark the bottom instead. Enabling the `Inverted` setting records the numbering layout accordingly for all racks, if the inverted rack is the first rack seen in `Rackview`.

Rack configuration settings in `cmsh`: Tab-completion suggestions for the `set` command in `rack` mode display the racks available for configuration. Tab-completion suggestions can also display the configuration settings available for a specified rack, for example, `rack2`:

Example

```
[basecm11->rack]% set rack
rack1  rack2  rack3
[basecm11->rack]% set rack2
angle         height        model         partnumber    row           type          y-coordinate
building      inverted      name          revision      serialnumber  width
depth         location      notes         room          twin          x-coordinate
```

The configuration settings for a particular rack match with the parameters of figure 3.21 and are discussed in the text that follows figure 3.21.

Setting the values can be done as in this example:

Example

```
[basecm11->rack]% use rack2
[basecm11->rack[rack2]]% set room "skonk works"
[basecm11->rack*[rack2*]]% set x-coordinate 2
[basecm11->rack*[rack2*]]% set y-coordinate 0
[basecm11->rack*[rack2*]]% set inverted no
[basecm11->rack*[rack2*]]% commit
[basecm11->rack[rack2]]%
```

3.16.2 Assigning Devices To A Rack

Devices such as nodes, switches, and chassis, can be assigned to racks.

By default, no such devices are assigned to a rack.

Devices can be assigned to a particular rack and to a particular position within the rack as follows:

Assigning Devices To A Rack Using Base View

Using Base View, a device such as a node `node001` can be assigned to a rack using the navigation path (figure 3.22):

`Devices > All Devices > node001`  `> Settings > Rack +`

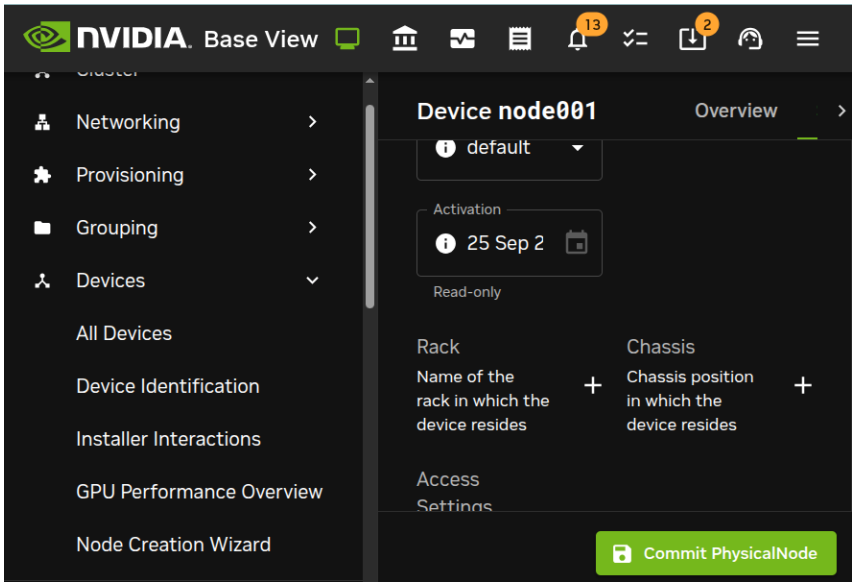


Figure 3.22: Rack assignment using Base View

Assigning Devices To A Rack Using `cmsh`

- In `device` mode, nodes can be assigned to a position in a rack. For example:
  - `node001` to position 1
  - `node002` to position 2
  - `node003` to position 3

  A looping instruction to do this in an existing rack `rack2` is something similar to:

```
root@basecm11:~# for i in {001..003} ; do cmsh -c "device use node$i; get hostname; set rack rack2; \
get rack; rackposition; set deviceposition $i; get deviceposition; commit"; done
```

- The `addrackposition` command fills devices into a rack object more implicitly. By default the command fills devices into the first free slots on the rack.

```
root@basecm11:~# cmsh-c "device; addrackposition -n node001.node010 --position 1 rack2
```

  The first free `deviceposition` value that it uses can be set with the `--position` option. Forcing a device into the the same position as another device is also possible, with the `--force` option.

  More options can be found using the `help` text for the command.

```
[basecm11->device]% help addrackposition
Name:
          addrackposition - Add a rack position information to one or more nodes

...
Examples:
          addrackposition -n node001..node010 rack1
          addrackposition -n node001..node010 --height 2 --position 4 rack rack2
```

- The `rackposition` submode allows individual nodes to be managed directly:

```
[basecm11->device[node002]->rackposition]% show
Parameter                      Value
------------------------------ ------------------------------------------------
Rack                           rack2
Revision
Device position                2
Device height                  1
Tray ID
Tray name
```

A similar submode to `rackposition` is `chassisposition` (section 3.16.4).

The Convention Of The Top Of The Device Being Its Position

Since rack manufacturers usually number their racks from top to bottom, the position of a device in a rack (using the parameter `Position` in Base View, and the parameter `deviceposition` in `cmsh`) is always taken to be where the top of the device is located. This is the convention followed even for the less usual case where the rack numbering is from bottom to top.

A position on a rack is 1U of space. Most devices have a height that fits in that 1U, so that the top of the device is located at the same position as the bottom of the device, and no confusion is possible. The administrator should however be aware that for any device that is greater than 1U in height such as, for example, a blade enclosure chassis (section 3.16.3), the convention means that it is the position of the top of the device that is where the device is considered to be. The position of the bottom of the device is ignored.

3.16.3 Assigning Devices To A Chassis

A Chassis As A Physical Part Of A Cluster

In a cluster, several local nodes may be grouped together physically into a chassis. This is common for clusters using blade systems. Clusters made up of blade systems use less space, less hardware, and less electrical power than non-blade clusters with the same computing power. In blade systems, the blades are the nodes, and the chassis is the blade enclosure.

A blade enclosure chassis is typically 6 to 10U in size, and the node density for server blades is typically 2 blades per unit with 2014 technology.

Chassis Configuration And Node Assignment

Chassis in Base View: The `Chassis` pane can be added in Base View using the navigation path

```
Devices > All Devices > + Add > Chassis
```

This opens a `Chassis` pane where a new chassis object can be configured. Properties such as name, MAC address, rack location, can be set and saved (figure 3.23).

An existing chassis object, for example `chassis01`, can be edited by selecting it from the navigation path `Devices > All Devices > chassis01`

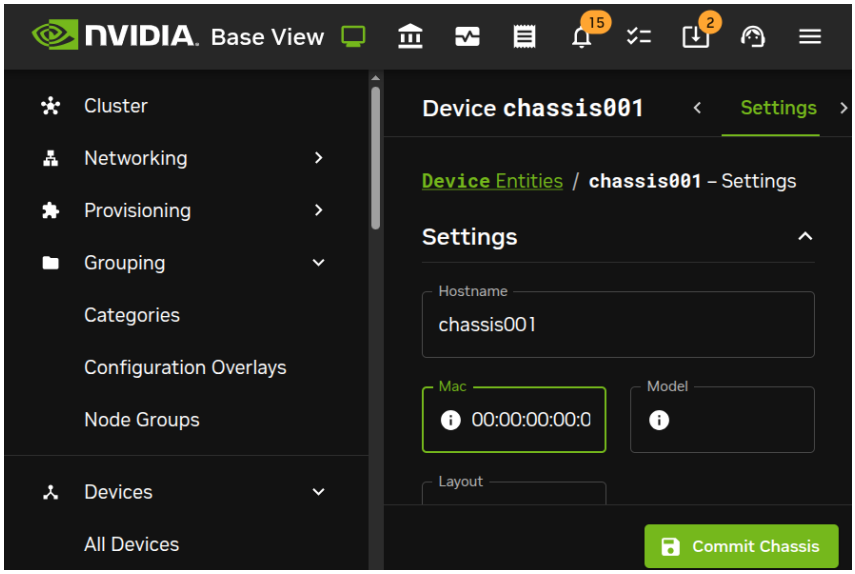


Figure 3.23: Base View chassis configuration

The options that can be set within the `Chassis` pane include the following:

- `Hostname`: a name that can be assigned to the chassis operating system
- `Mac`: the MAC address of the chassis
- `Model`: the hardware model name
- `Rack`: the rack in which the chassis is placed
- `Layout`: how the nodes in a chassis are laid out visually.
- `Network`: which network the chassis is attached to.
- `Access Settings`: Opens up a pane to allow a user name and password to be set to access the chassis operating system, as well as the REST API port.
- `Power control`, `Custom power script`, `Custom power script argument`: power-related items for the chassis.
- `BMC Settings`: Baseboard Management Controller settings for the chassis.
- `Userdefined1`, `Userdefined2`: administrator-defined variables that can be used by CMDaemon.

Basic chassis configuration and node assignment with `cmsh`: The `chassis` type in `cmsh` allows configuration related to a particular chassis. Tab-completion suggestions for a selected chassis with the `set` command show possible parameters that may be set:

Example

```
[basecm11->device]% add chassis chassis1
[basecm11->device*[chassis1*]]% set
accesssettings              defaultgatewaymetric        partition                   serialnumber
bmcsettings                 hostname                    partnumber                  supportsgnss
chassis                     interfaces                  powercontrol                switchports
custompingscript            layout                      powerdistributionunits      userdefined1
custompingscriptargument    mac                         prometheusmetricforwarders  userdefined2
```

```
custompowerscript           managementnetwork          provisioninginterface      userdefinedresources
custompowerscriptargument   model                      rack
defaultgateway              notes                      revision
```

Whether the suggested parameters are actually supported depends on the chassis hardware. For example, if the chassis has no power control or network interface of its own, then the `powercontrol` value and `mac` address settings may be set, but they cannot function.

The positioning parameters of the chassis within an already-defined rack, `rack2`, can be set as follows with `cmsh`:

Example

```
[basecm11->device[node001]]% rackposition
[basecm11->device*[node001*]->rackposition*]% set rack rack2
[basecm11->device*[node001*]->rackposition*]% set deviceposition 1; set deviceheight 6
[basecm11->device*[node001*]->rackposition*]% commit
```

The members of the chassis can be set as follows with `cmsh`:

Example

```
[basecm11->device[chassis1]]% append members basecm11 node001..node005
[basecm11->device*[chassis1*]]% commit
```

3.16.4 The `chassisposition` Submode

Section 3.16.2 described positioning attributes available in `device` mode, within the `rackposition` submode, for a device within a rack.

This section, describes positioning attributes available in `device` mode, within the `chassisposition` submode, for a device within a chassis.

Attributes include the chassis that the device belongs to, the slot, position, and width of the device within that chassis.

Example

```
[basecm11->device]% chassisposition node001
[basecm11->device[node001]->chassisposition]% show
Parameter                      Value
------------------------------ ------------------------------------------------
Chassis
Slot
Position                       1
Width                          1
```

3.16.5 Power Management Of Racks

Section 4.2 of this manual, describes the power operations and power status responses for devices in general, while section 4.9.4 of the *NVIDIA Mission Control Manual* goes into some more detail for particular rack operations.

3.17 Configuring GPU Settings

3.17.1 GPUs And GPU Units

GPUs (Graphics Processing Units) are processors that are heavily optimized for executing certain types of parallel processing tasks. GPUs were originally used for rendering graphics, and one GPU typically has hundreds of cores. When used for general processing, they are sometimes called General Processing GPUs, or GPGPUs. For convenience, the "GP" prefix for General Processing is not used in this manual.

A GPU is typically placed on a PCIe card. GPUs can be physically inside the node that uses them, or they can be physically external to the node that uses them. As far as the operating system on the node making use of the physically external GPUs is concerned, the GPUs are internal to the node.

If the GPUs are physically external to the node, then they are typically in a *GPU unit*. A GPU unit is a chassis that hosts only GPUs. It is typically able to provide GPU access to several nodes, usually via PCIe extender connections. This ability means that external GPUs typically require more configuration than internal GPUs. GPU units are not covered in this manual because they are not very popular due to their greater cost and slowness.

Configuring GPU settings for GPUs—that is, for devices internal to a node—is covered next.

3.17.2 Configuring GPU Settings

The `gpusettings` Submode In `cmsh`

In `cmsh`, GPUs can be configured for a specified node via `device` mode.

Going into the `gpusettings` submode for that node then allows a type of GPU to be set, from the `amd` or `nvidia` types, and a range to be specified for the GPU slots for that particular node:

Example

```
[root@basecm11 ~]# cmsh
[basecm11]% device use node001
[basecm11->device[node001]]% gpusettings
[basecm11->device[node001]->gpusettings]% add nvidia 1-3 ; commit
```

The range can be specified as

- a single number, for a particular slot, for example: `3`
- a range, for a range of slots, for example: `0-2`
- all, for all GPU slots on that node, using:

  `all`

  or

  `*`

GPUs can also be configured for a specified category via `category` mode. For example, using the category `default`, then entering into the `gpusettings` submode allows a type (`nvidia` or `amd`) and a range to be set for the range of GPUs:

Example

```
[root@basecm11 ~]# cmsh
[basecm11]% category use default
[basecm11->category[default]]% gpusettings
[basecm11->category[default]->gpusettings]% list
GPU range (key)  Power limit  ECC mode     Compute mode  Clock speeds
---------------- ------------ ------------ ------------- -------------
[basecm11->category[default]->gpusettings]% add nvidia 1-3 ; commit
[basecm11->category[default]->gpusettings[nvidia:1-3]]% show
Parameter                      Value
------------------------------ ---------------------------------
Clock speeds
Clock sync boost mode
Compute mode
ECC mode
...
```

As usual, GPU settings for a node override those for a category (section 2.1.3).

GPU Settings With NVIDIA GPUs

The installation of the NVIDIA GPU software driver packages is covered in section 9.1 of the *Installation Manual*. It should be noted that the `cuda-dcgm` package must be installed to access NVIDIA GPU metrics.

The present section is about configuring NVIDIA GPUs in BCM. The driver itself does not necessarily have to be in place for the configuration to be done, although the configuration only becomes active when the driver is installed.

After a GPU type has been set, the following NVIDIA GPU settings may be specified, if supported, from within the `gpusettings` submode:

- `clockspeeds`: The pair of clock speeds (frequency in MHz) to be set for this parameter can be selected from the list of available speeds. The available speeds can be seen by running the `status` command. The values are specified in the form: *<number for GPU processor>,<number for memory>*
- `clocksyncboostmode`: GPU boosting. Exceed the maximum core and memory clock speeds if it is safe. Choices are:
  - `enabled`
  - `disabled`
- `computemode`: Contexts can be computed with the following values for this mode:
  - `Default`: Multiple contexts are allowed
  - `Exclusive thread`: Only one context is allowed per device, usable from one thread at a time
  - `Exclusive process`: Only one context is allowed per device, usable from multiple threads at a time. This mode option is valid for CUDA 4.0 and later. Earlier CUDA versions ran only in this mode.
  - `Prohibited`: No contexts are allowed per device
- `eccmode`: Sets the ECC bit error check, with:
  - `enabled`
  - `disabled`

  When ECC is enabled:
  - Single bit errors are detected, using the `EccSBitGPU` metric (page 993), and corrected automatically.
  - Double bit errors are also detected, using the `EccDBitGPU` metric (page 993), but cannot be corrected.
- `GPU range`: range values can be set as follows:
  - `all`: The GPU settings apply to all GPUs on the node.
  - *<number>*: The GPU settings apply to an individual GPU, for example: `1`
  - *<number range>*: The GPU settings apply to a range of GPUs, for example: `1,3-5`
- `powerlimit`: The administrator-defined upper power limit for the GPU. Only valid if `powermode` is `Supported`.
  - `min`: The minimum upper power limit that the hardware supports.
  - `max`: The maximum upper power limit that the hardware supports.
  - *<number>*: An arbitrary upper power limit, specified as a number between `min` and `max`
  - `default`: Upper power limit based on the default hardware value.

If no value is specified for a GPU setting, then the hardware default is used.

The updatedevicepowerconfig Command For Controlling Power Consumption

Above the gpusettings submode, within device mode, nodes can have their power consumption limited per specified GPU or CPU:

Without arguments, the updatedevicepowerconfig command shows the current status for GPUs:

Example

```
[basecm11->device[node001]]% updatedevicepowerconfig
Node         Kind         Device       Power limit
------------ ------------ ------------ ----------------
node001      GPU          GPU0         350 W
node001      GPU          GPU1         300 W
```

In the preceding example, the only node with a GPU is node001. The same result is therefore shown if the command is run as updatedevicepowerconfig -n node001.

The kind of processing unit that can have its power configuration examined and set can be either a GPU or CPU. A CPU status command can be specified by using the --kind option:

Example

```
[basecm11->device[node001]]% updatedevicepowerconfig -n node001 --kind CPU
Node         Kind         Device       Power limit
------------ ------------ ------------ ----------------
node001      CPU          CPU0         200 W
node001      CPU          CPU1         200 W
```

The first and second GPUs on node001 can have their power limits set to the same value with a range syntax and the --power-limit option:

```
[basecm11->device]% updatedevicepowerconfig node001:0-1  --power-limit 330
Node         Kind         Device       Power limit
------------ ------------ ------------ ----------------
node001      GPU          GPU0         330 W
node001      GPU          GPU1         330 W
```

For the first GPU on node001, and for all CPUs on node002 the power limit, in watts, is set to specified values:

```
[basecm11->device]% updatedevicepowerconfig node001:0:300 node002:CPU*:150
Node         Kind         Device       Power limit
------------ ------------ ------------ ----------------
node001      GPU          GPU0         300 W
node001      GPU          GPU1         300 W
node002      CPU          CPU0         150 W
node002      CPU          CPU1         150 W
```

A dry-run option --dry-run shows the effect on both the GPUs without execution.

```
[basecm11->device]% updatedevicepowerconfig node001:0:300 node002:CPU*:150
Node         Kind         Device       Power limit
------------ ------------ ------------ ----------------
node001      GPU          GPU0         300 W
node001      GPU          GPU1         300 W
node002      CPU          CPU0         150 W
node002      CPU          CPU1         150 W

Setting the power limit in  both  gpusettings values and  updatedevicepowerconfigs may cause conflict.
```

GPU Settings With AMD GPUs

GPU settings for AMD Radeon GPUs are accessed via `cmsh` in the same way as NVIDIA GPU settings. The AMD GPU setting parameters do differ from the NVIDIA ones.

The AMD GPUs supported are Radeon cards. A list of cards and operating systems compatible with the Linux driver used is at `https://support.amd.com/en-us/kb-articles/Pages/Radeon-Software-for-Linux-Release-Notes.aspx`

AMD GPU driver installation is described in section 7.4 of the *Installation Manual*.

The Radeon Instinct MI25 shows the following settings in Ubuntu 16_06 running a Linux 4.4.0-72-generic kernel:

Example

```
[basecm11->device[node001]->gpusettings]% list
Type GPU range Info
---- --------- ------------------
AMD  0         PowerPlay: manual
[basecm11->device[node001]->gpusettings]% use amd:0
[basecm11->device[node001]->gpusettings[amd:0]]% show
Parameter                      Value
------------------------------ ----------------------------
Activity threshold             1
Fan speed                      255
GPU clock level                5
GPU range                      0
Hysteresis down                0
Hysteresis up                  0
Info                           PowerPlay: manual
Memory clock level             3
Minimum GPU clock              0
Minimum memory clock           0
Overdrive percentage           1
PowerPlay mode                 manual
Revision
Type                           AMD
```

The possible values here are:

- `activitythreshold`: Percent GPU usage at a clock level that is required before clock levels change. From 0 to 100.
- `fanspeed`: Maximum fan speed. From 0 to 255
- `gpuclocklevel`: GPU clock level setting. From 0 to 7.
- `gpurange`: The slots used.
- `hysteresisdown`: Delay in milliseconds before a clock level decrease is carried out.
- `hysteresisup`: Delay in milliseconds before a clock level increase is carried out.
- `info`: A compact informative line about the GPU status.
- `memoryclocklevel`: Memory clock speed setting. From 0-3. Other cards can show other values.
- `minimumgpuclock`: Minimum clock frequency for GPU, in MHz. The kernel only allows certain values. Supported values can be seen using the `status` command.

- `minimummemoryclock`: Minimum clock frequency for the memory, in MHz. The kernel only allows certain values. Supported values can be seen using the status command.
- `overdrivepercentage`: Percent overclocking. From 0 to 20%
- `powerplaymode`: Decides how the performance level power setting should be implemented.
  - `high`: keep performance high, regardless of GPU workload
  - `low`: keep performance low, regardless of GPU workload
  - `auto`: Switch clock rates according to GPU workload
  - `manual`: Use the memory clock level and GPU clock values.

The `status` command displays supported clock frequencies (some values ellipsized):

Example

```
[basecm11->device[node001]->gpusettings[amd:0]]% status
Index Name                  Property     Value        Supported
----- --------------------- ------------ ------------ -----------------------------------
0     Radeon Instinct MI25  Clock        1399Mhz      852Mhz, 991Mhz, ..., 1440Mhz, 1515Mhz
0     Radeon Instinct MI25  Memory       945Mhz       167Mhz, 500Mhz, 800Mhz, 945Mhz
```

GPU Settings In Base View

In Base View, `GPU settings` can be accessed within the settings options for a category or a device. This brings up a GPU settings list.

GPU settings list in Base View: A `GPU Settings list` pane can be opened up in Base View for a regular node, for example `node001`, with the navigation path:

`Devices > All Devices > Nodes`  `> Settings > GPU Settings` 

Similarly, a `GPU Settings list` pane can be opened up in Base View for nodes in a category, for example `gpunodes`, with the navigation path:

`Grouping > Categories[gpunodes]` `> Settings > GPU Settings` 

Within the `GPU Settings list` pane:

- a new GPU settings item can be added with the `+Add` button, and then clicking on either the `AMD` or the `NVIDIA` item in the popup. This opens an `AMDGPUSettings` pane or an `NvidiaGPUSettings` pane, where GPU configuration can be carried out and saved (figure 3.24).

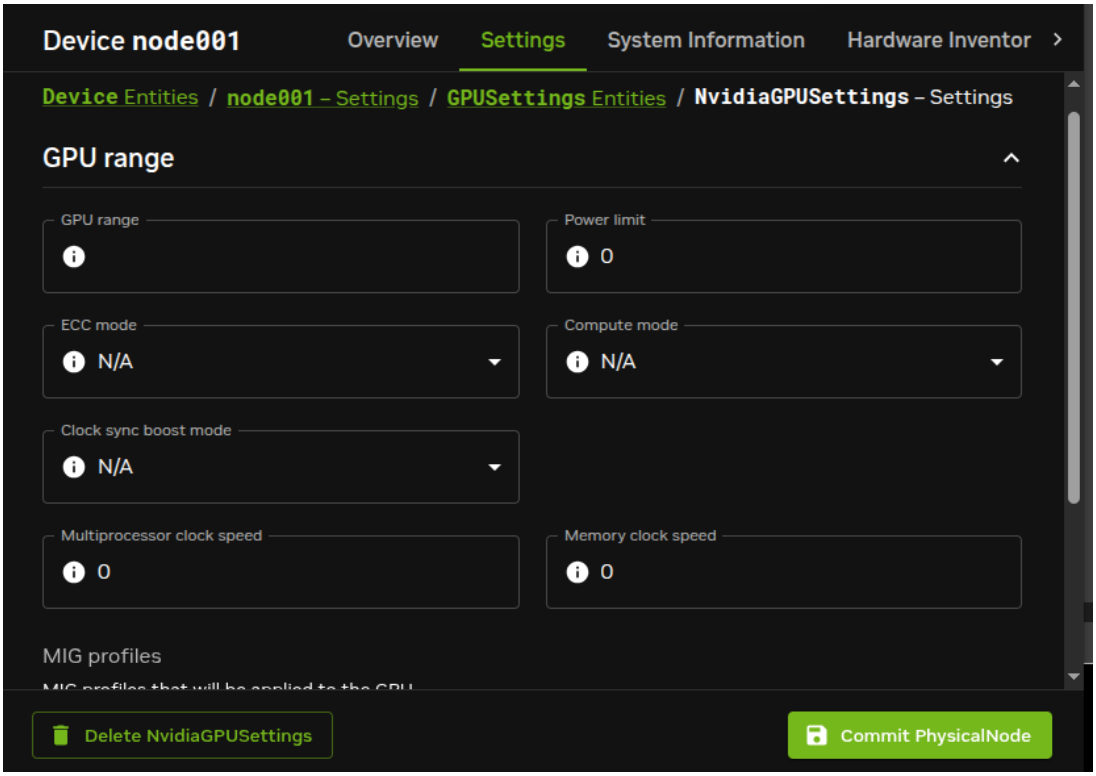


Figure 3.24: GPU settings window for a node

- an existing GPU settings item can be edited with the option, which opens up the GPU Settings pane where GPU configuration can be managed.

GPU Configuration For HPC Workload Managers

Slurm GPU configuration via direct `slurm.conf` changes: To configure NVIDIA GPUs for Slurm, changes are made in `slurm.conf` when `cm-wlm-setup` configures GPUs for Slurm (section 7.3).

Changes made are kept in the `AUTOGENERATED` section and can be worked out by checking the difference between the `slurm.conf.template` file and the actual `slurm.conf` file. Changes made include defining the GresTypes `gpu` and `mps`, and setting GPU plugins that allow Slurm generic resources to work.

The configured gres options can be seen by running `sbatch --gres=help`:

Example

```
[fred@basecm11 ~]$ sbatch --gres=help
Valid gres options are:
gpu[[:type]:count]
mps[[:type]:count]
```

This means that a GPU can be requested in a job script with the Slurm gres option:

```
#SBATCH --gres=gpu:1
```

Similarly, MPS resources (`https://slurm.schedmd.com/gres.html#MPS_Management`) can be requested with:

```
#SBATCH --gres=mps:100
```

If adding new parameters manually, care must be taken to avoid duplication of parameters already in the file, because `slurmd` is unlikely to work properly with duplicated parameters.

The Slurm client role can be configured at configuration overlay, category, or node level. If configuring the Slurm client role for GPU gres resources manually, then each GPU can be configured within the role:

Example

```
[basecm11->configurationoverlay]% list
Name (key)           Priority   All head nodes Nodes            Categories       Roles
-------------------- ---------- -------------- ---------------- ---------------- ----------------
slurm-accounting     500        yes                                              slurmaccounting
slurm-client         500        no                               default          slurmclient
slurm-server         500        yes                                              slurmserver
slurm-submit         500        no                               default          slurmsubmit
wlm-headnode-submit  600        yes                                              slurmsubmit
[basecm11->configurationoverlay]% use slurm-client
[basecm11->configurationoverlay[slurm-client]]% roles
[basecm11->configurationoverlay[slurm-client]->roles]% use slurmclient
[basecm11->configurationoverlay[slurm-client]->roles[slurmclient]]% genericresources
[basecm11->configurationoverlay[slurm-client]->roles[slurmclient]->genericresources]%
[basecm11->...->roles[slurmclient]->genericresources]% add gpu0
[basecm11->...->roles*[slurmclient*]->genericresources*[gpu0*]]% set name gpu
[basecm11->...->roles*[slurmclient*]->genericresources*[gpu0*]]% set file /dev/nvidia0
[basecm11->...->roles*[slurmclient*]->genericresources*[gpu0*]]% commit
[basecm11->...->roles[slurmclient]->genericresources[gpu0]]%
    (Repeat similar settings for the other GPUs, gpu1...gpu7)
[basecm11->...->roles[slurmclient]->genericresources]% list
  Alias (key) Name     Type     Count    File
  ----------- -------- -------- -------- ----------------
  gpu0        gpu                        /dev/nvidia0
  gpu1        gpu                        /dev/nvidia1
  gpu2        gpu                        /dev/nvidia2
  gpu3        gpu                        /dev/nvidia3
  gpu4        gpu                        /dev/nvidia4
  gpu5        gpu                        /dev/nvidia5
  gpu6        gpu                        /dev/nvidia6
  gpu7        gpu                        /dev/nvidia7
[basecm11->configurationoverlay[slurm-client]->roles[slurmclient]->genericresources]%
```

By default, Slurm just allows a single job to be executed per node. To change this behavior, it is necessary to allow oversubscription. For example, to allow 8 jobs per node:

Example

```
[basecm11->wlm[slurm]]% jobqueue
[basecm11->wlm[slurm]->jobqueue]% use defq
[basecm11->wlm[slurm]->jobqueue[defq]]% set oversubscribe yes:8
[basecm11->wlm[slurm]->jobqueue*[defq*]]% commit
[basecm11->wlm[slurm]->jobqueue[defq]]%
```

Slurm GPU configuration via auto-detection: Instead of carrying out Slurm configuration by modifying `slurm.conf` by hand, it may be configured via auto-detection. More details on this are to be found starting at page 425.

LSF: Within LSF cluster configuration, GPU devices can be autodetected by setting the `gpuautoconfig` parameter to `yes`. In `cmsh` this can be carried out with:

Example

```
[basecm11->wlm[lsf]]% set gpuautoconfig yes
[basecm11->wlm*[lsf*]]%
```

The parameter can also be set during LSF configuration via `cm-wlm-setup` (figure 3.25):

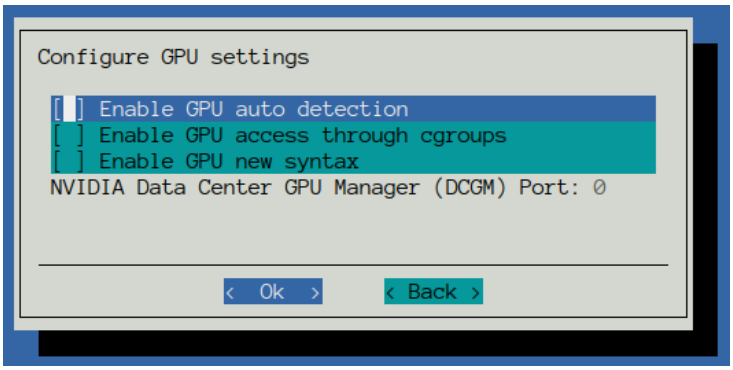


Figure 3.25: GPU settings screen for LSF in cm-wlm-setup

GPU resource enforcement can be configured for LSF as follows:

Example

```
[basecm11->wlm[lsf]->cgroups]% append resourceenforce gpu
[basecm11->wlm*[lsf*]->cgroups*]% commit
[basecm11->wlm[lsf]->cgroups]%
```

3.17.3 MIG Configuration

MIG configuration can be carried out for the cluster using the BCM MIG management as described in this section.

An alternative for MIG configuration is to not use the BCM MIG management tool (`cmsh`), and to instead use other MIG management tools, such as the DGX native `nvidia-migmanager.service`, or the GPU operator-provided `nvidia-mig-manager`. If non-BCM MIG management tools are used, then BCM leaves the MIG configuration alone. Using multiple MIG management tools simultaneously to configure MIG should not be done.

What Is MIG?

An Ampere NVIDIA GPU is a GPU based on the GA100 microarchitecture. It has compute capability 8.0, which means it can be configured into multiple logical GPU instances if it uses CUDA 11 and NVIDIA driver 450.80.02 or later.

This configuration of multiple logical GPU instances is called Multi-Instance GPU (MIG). The logical GPU instances are MIG devices, that are enabled by setting up the physical GPU to switch to MIG mode.

As a sanity check to see if MIG is supported: If the hardware and drivers are in place, then running the `nvidia-smi` command on the node with the physical GPU should display its MIG capability:

```
root@basecm11:~# ssh node001 "nvidia-smi" | grep MIG
|                               |                      |               MIG M. |
| MIG devices:                                                               |
| GPU  GI  CI  MIG |         Memory-Usage |        Vol|         Shared         |
```

GPU utilization information changes on enabling MIG: Once enabled, the full physical GPU is no longer available as a device, and GPU utilization metrics become unavailable by default.

GPU profiling metrics (section G.1.7) for the physical GPU can however still be enabled. For example, it can be carried out with `cmsh` as follows, for a GPU on `node001`:

Example

```
basecm11->device[node001]]% gpuprofiling show
Hostname    GPU      Major ID     Minor ID     Field ID     Metric                          Watched
----------- -------- ------------ ------------ ------------ ------------------------------ ----------
node001     0        0            1            1002         gpu_profiling_sm_active         no
node001     0        0            1            1003         gpu_profiling_sm_occupancy      no
...
basecm11->device[node001]]% gpuprofiling watch 1003
Hostname    GPU      Major ID     Minor ID     Field ID     Metric                          Watched
----------- -------- ------------ ------------ ------------ ------------------------------ ----------
...
node001     0        0            1            1003         gpu_profiling_sm_occupancy      yes
...

[basecm11->device[node001]]% metrics | grep gpu_profiling
Metric  gpu_profiling_sm_occupancy       gpu0                    GPU       GPUSampler
```

Enabling physical GPU profiling after MIG enablement should be done with caution, because:

- it may affect the performance
- newer drivers may support MIG profiling, which may be confusing

Overview Of MIG Concepts And Terminology

The logical GPU instances are composed of *slices* of GPU resources. Slices are the smallest fraction possible of the resource that can be allocated in a logical GPU instance. Thus:

- memory slice: this is the smallest fraction of the memory of the physical GPU that can be allocated to the GPU instance. For the Ampere architecture this is $1/8^{\text{th}}$ of the total physical GPU memory.
- SM slice: this is the smallest fraction of the streaming multiprocessors (SMs) on the GPU that can be allocated as a logical SM. An SM is composed of multiple cores (streaming processors). For the Ampere architecture an SM slice is $1/7^{\text{th}}$ of the total physical GPU SMs.
- GPU slice: this is the smallest fraction of the physical GPU that has a single GPU memory slice and a single GPU SM slice. A maximum of 7 GPU slices can be specified from the original physical GPU.

The preceding fractional slices can be combined in various mixes to compose a logical `GPU instance`:

- GPU instance: a combination of GPU slices and GPU engines. GPU engines are hardware components that execute other work on the GPU, and can be encoders/decoders (NVENCs/NVDECs), shortcut connectors (CE (copy engine) for DMA), etc.
- compute instance: a part of a GPU instance. It consists of a subset of the parent GPU instance's SM slices and other GPU engines (DMAs, NVENCs ...). The compute instances can share memory and GPU engines with other compute instances within their GPU instance.

Further details on the terminology and how slices can be allocated to GPU instances are given in the NVIDIA documentation at `https://docs.nvidia.com/datacenter/tesla/mig-user-guide/#concepts`.

A use case for creating several GPU instances from a full physical GPU is when allocation of the full physical GPU is wasteful.

For example, if a full physical GPU is allocated to a job, but the job only uses a fraction of the full set of GPU cores, then the allocation is wasteful, because no other job can then be processed on the remaining idle cores. Instead, if the physical GPU is split into several instances and the job allocated to an instance with a closer match in resources, then it means that other GPU instances are still available for processing other jobs.

When configuring GPU instances for the cluster, the administrator typically allocates all available slices to all the GPU instances that are being configured. Leaving a slice unallocated means that that slice cannot be available to jobs, and it means that the physical resources of that slice are lying idle. For example, if some of the 7 SM slices from the physical GPU are not used, then they are wasted as their processors are never available to jobs, and so that slice stays idle.

Cluster management of GPU instances is described in the following sections.

MIG Status Of Physical GPUs

The MIG status on a GPU can be viewed with the `mig status` command. For example, the following shows 8 physical GPUs on `node001` that have not yet become MIG enabled:

Example

```
[basecm11->device[node001]]% mig status
Node             GPU      Active                          Pending
---------------- -------- ------------------------------- -------------------------------
node001          0        no                              no
node001          1        no                              no
node001          2        no                              no
node001          3        no                              no
node001          4        no                              no
node001          5        no                              no
node001          6        no                              no
node001          7        no                              no
```

MIG `enable` And `disable` Options To Set Up Pending States For The Physical GPUs

If the `enable` or `disable` options are run, then by default all the physical GPUs are set to a pending state for enabled (`yes`) and disabled (`no`) respectively:

Example

```
[basecm11->device[node001]]% mig enable
Node             GPU      Active                          Pending
---------------- -------- ------------------------------- -------------------------------
node001          0        no                              yes
node001          1        no                              yes
node001          2        no                              yes
node001          3        no                              yes
node001          4        no                              yes
node001          5        no                              yes
node001          6        no                              yes
node001          7        no                              yes
```

Individual physical GPUs can also be set to a pending state of enabled or disabled, following the node list syntax (section 2.5.5):

```
[basecm11->device[node001]]% mig disable 3,5-7
Node             GPU      Active                          Pending
```

```
---------------- -------- ------------------------------ ------------------------------
node001          0        no                             yes
node001          1        no                             yes
node001          2        no                             yes
node001          3        no                             no
node001          4        no                             yes
node001          5        no                             no
node001          6        no                             no
node001          7        no                             no
```

Rebooting The MIG Instances To Activate/Deactivate Instances According To Pending State Settings

The pending states only become active after the node is rebooted for A100 GPUs:

Example

```
[basecm11->device[node001]]% reboot
Reboot in progress for: node001
...
node001 [   UP   ]
[basecm11->device[node001]]% mig status

Node             GPU      Active                         Pending
---------------- -------- ------------------------------ ------------------------------
node001          0        yes                            yes
node001          1        yes                            yes
node001          2        yes                            yes
node001          3        no                             no
node001          4        yes                            yes
node001          5        no                             no
node001          6        no                             no
node001          7        no                             no
```

The effect of `enable` and `disable` persists after reboots, until the pending value changes once more. For H100 GPUs and other more modern GPUs, the changes do not require a reboot.

The MIG Profiles

Listing MIG profiles: The full list of the existing available MIG profiles for each physical GPU is displayed with the `mig profiles` command for the node. If there are 7 physical GPUs at the node, then the listing might look something like:

Example

```
[basecm11->device[node001]]% mig profiles
Node             GPU      ID       Name       Instances Memory
---------------- -------- -------- ---------- --------- --------
node001          0        1        1g.5gb     7         4.7GiB
node001          0        1        1g.5gb+me  1         4.7GiB
node001          0        2        2g.10gb    3         9.7GiB
node001          0        3        3g.20gb    1         19.6GiB
node001          0        4        4g.20gb    1         19.6GiB
node001          0        7        7g.40gb    1         39GiB
...
node001          7        2        2g.10gb    3         9.7GiB
node001          7        3        3g.20gb    1         19.6GiB
node001          7        4        4g.20gb    1         19.6GiB
node001          7        7        7g.40gb    1         39GiB
```

A list for physical GPU 1 can be displayed with:

Example

```
[basecm11->device[node001]]% mig profiles 1
Node             GPU      ID       Name       Instances Memory
---------------- -------- -------- ---------- --------- --------
node001          1        1        1g.5gb     7         4.7GiB
node001          1        1        1g.5gb+me  1         4.7GiB
node001          1        2        2g.10gb    3         9.7GiB
node001          1        3        3g.20gb    1         19.6GiB
node001          1        4        4g.20gb    1         19.6GiB
node001          1        7        7g.40gb    1         39GiB
```

MIG profiles naming convention: The naming format for the profile takes the form

<number of GPU slices in the physical GPU>`g.`*<memory for slice in GB>*`gb`

The `+me` suffix implies media extensions being active. The number of GPU instances, the number of GPU slices used, and the memory used by the instance can thus be worked out from the name.

For example:

- `1g.5gb` implies that the size of the GPU slice used for the instances is 1. 4.7GiB of memory is used by each of the 7 GPU instances,
- `2g.10gb` implies that the size of the GPU slice used for the instances is 2. 9.7GiB of memory is used by each of the 3 GPU instances.

Creating GPU instances, by setting the MIG profile for a GPU: The MIG profile is an attribute that can be set within the GPU settings for its physical GPU. This can be done after having set up a CMDaemon entity for a physical GPU 0 (section 3.17.2):

Example

```
[basecm11->device[node001]]% gpusettings
[basecm11->device*[node001]->gpusettings]% list
Type   GPU range Info
------ --------- --------
Nvidia 0         default
[basecm11->device[node001]->gpusettings]% use nvidia:0
[basecm11->device[node001]->gpusettings[nvidia:0]]% show
Parameter                      Value
------------------------------ ------------------------------------------------
...
MIG profiles
```

- **A simple existing set of profiles** with 7 GPU instances with 1 GPU slice each, and 5 GB of memory for each slice can be specified with:

  Example

  ```
  [basecm11->device[node001]->gpusettings[nvidia:0]]% set migprofiles 1g.5gb; commit
  [basecm11->device[node001]->gpusettings[nvidia:0]]% show
  Parameter                      Value
  ------------------------------ ------------------------------------------------
  ...
  MIG profiles                   1g.5gb
  ```

Setting the profiles, and carrying out the `commit` configures the GPU instances, but does not yet apply them:

Example

```
[basecm11->device[node001]->gpusettings[nvidia:0]]% ..;..
[basecm11->device[node001]]% mig show
Node             GPU      MIG      Instance Name     Profile  Start    Size
---------------- -------- -------- -------- -------- -------- -------- --------
```

Applying the profile deploys the configuration, and shows the configuration:

Example

```
[basecm11->device[node001]]% mig apply
Node             GPU      MIG      Instance Name     Profile  Start    Size
---------------- -------- -------- -------- -------- -------- -------- --------
node001          0        13       -        1g.5gb   19       6        1
node001          0        13       0        1g.5gb   0        0        1
```

In the preceding, 1 GPU instance has been deployed, with 1 compute instance slice using 5GB. The instance with the - represents the hosting GPU instance, while the subsequent row represents the compute instance.

- **Setting a profile with** `mig apply --profile` is an alternative to setting it within `gpusettings`. However, only a profile set within `gpusettings` is persistent. The profile set with the `--profile` option is lost if its node reboots. Multiple profiles can be set using comma-separation (instead of using space-separation). Using multiple `--profile` options is also possible.

Example

```
[basecm11->device[node001]]% mig apply --profile 1g.5gb,2g.10gb
Node             GPU      MIG      Instance Name     Profile  Start    Size
---------------- -------- -------- -------- -------- -------- -------- --------
node001           0        5        -        2g.10gb  14       4        2
node001           0        5        0        2g.10gb  1        0        2
node001           0        13       -        1g.5gb   19       6        1
node001           0        13       0        1g.5gb   0        0        1
[basecm11->device[node001]]% mig apply --profile 1g.5gb --profile 2g.10gb
Node             GPU      MIG      Instance Name     Profile  Start    Size
---------------- -------- -------- -------- -------- -------- -------- --------
node001           0        5        -        2g.10gb  14       4        2
node001           0        5        0        2g.10gb  1        0        2
node001           0        13       -        1g.5gb   19       6        1
node001           0        13       0        1g.5gb   0        0        1
```

- **Multiple GPU instances** can be specified, if the GPU allows it, using a <*number*>* prefix syntax. So, to deploy 7 GPU instances, each hosting 1 compute instance with 5gb slices, the specification can be:

Example

```
[basecm11->device[node001]]% gpusettings; use nvidia:0
[basecm11->device[node001]->gpusettings[nvidia:0]]% set migprofiles 7*1g.5gb; commit
[basecm11->device[node001]->gpusettings[nvidia:0]]% ..;..
[basecm11->device[node001]]% mig apply
Node            GPU      MIG      Instance Name     Profile  Start    Size
--------------- -------- -------- -------- -------- -------- -------- --------
node001         0        7        -        1g.5gb   19       0        1
node001         0        7        0        1g.5gb   0        0        1
node001         0        8        -        1g.5gb   19       1        1
node001         0        8        0        1g.5gb   0        0        1
node001         0        9        -        1g.5gb   19       2        1
node001         0        9        0        1g.5gb   0        0        1
node001         0        10       0        1g.5gb   19       3        1
node001         0        10       0        1g.5gb   0        0        1
node001         0        11       -        1g.5gb   19       4        1
node001         0        11       0        1g.5gb   0        0        1
node001         0        12       -        1g.5gb   19       5        1
node001         0        12       0        1g.5gb   0        0        1
node001         0        13       -        1g.5gb   19       6        1
node001         0        13       0        1g.5gb   0        0        1
```

- **GPU slices** can be implied by default by the profile, and subsets of these slices can be specified explicitly.

  To deploy 1 GPU instance, with 7 GPU slices of compute instance resources, the specification can be:

  Example

```
[basecm11->device[node001]]% gpusettings; use nvidia:0
[basecm11->device[node001]->gpusettings[nvidia:0]]% set migprofiles 7g.40gb; commit
[basecm11->device[node001]->gpusettings[nvidia:0]]% ..;..
[basecm11->device[node001]]% mig apply
Node            GPU      MIG      Instance Name     Profile  Start    Size
--------------- -------- -------- -------- -------- -------- -------- --------
node001         0        0        -        7g.40gb  0        0        8
node001         0        0        0        7g.40gb  4        0        7
```

  If a profile with a more than 1 GPU slice is chosen, then GPU slice subsets can be set up via the following syntax:

  Example

```
[basecm11->device[node001]]% gpusettings; use nvidia:0
[basecm11->device[node001]->gpusettings[nvidia:0]]% set migprofiles 2g.10gb; commit
[basecm11->device[node001]->gpusettings[nvidia:0]]% ..;..
[basecm11->device[node001]]% mig apply
Node            GPU      MIG      Instance Name     Profile  Start    Size
--------------- -------- -------- -------- -------- -------- -------- --------
node001         0        5        -        2g.10gb  14       4        2
node001         0        5        0        2g.10gb  1        0        2
```

- **GPU slices use a colon syntax** to explicitly specify subsets of GPU slices.

  The configuration specification `2g.10gb` can also be specified as `2g.10gb:1`, where that `:1` indicates the number of GPU slices for the compute instance, counting from zero. That means the compute instance has 2 GPU slices. The resulting configuration is exactly the same as `2g.10g`.

If the specification `2g.10gb:0` is used instead, then the compute instance ends up looking like:

Example

```
[basecm11->device[node001]]% gpusettings; use nvidia:0
[basecm11->device[node001]->gpusettings[nvidia:0]]% set migprofiles 2g.10gb:0; commit
[basecm11->device[node001]->gpusettings[nvidia:0]]% ..;..
[basecm11->device[node001]]% mig apply
Node             GPU      MIG      Instance Name        Profile  Start    Size
---------------- -------- -------- -------- ----------- -------- -------- --------
node001          0        5        -        2g.10gb     14       4        2
node001          0        5        0        1c.2g.10gb  0        0        1
```

Here the `1c` indicates 1 GPU slice (here it is counting from 1).

Adding another slice to a separate compute instance within the same GPU instance can be specified with:

Example

```
[basecm11->device[node001]]% gpusettings; use nvidia:0
[basecm11->device[node001]->gpusettings[nvidia:0]]% set migprofiles 2g.10gb:0:0; commit
[basecm11->device[node001]->gpusettings[nvidia:0]]% ..;..
[basecm11->device[node001]]% mig apply
Node             GPU      MIG      Instance Name        Profile  Start    Size
---------------- -------- -------- -------- ----------- -------- -------- --------
node001          0        5        -        2g.10gb     14       4        2
node001          0        5        0        1c.2g.10gb  0        0        1
node001          0        5        1        1c.2g.10gb  0        1        1
```

In the preceding `set migprofiles` command, the specification

`2g.10gb:0:0`

can alternatively be expanded out and written in the form:

`2g.10gb:1c.2g.10gb:1c.2g.10gb`

for more clarity, at the expense of more typing.

For a general profile that allows N GPU slices for an instance (with $N \leq 7$), the mapping for the colon syntax takes the form:

compact colon form	expanded form
`:0`	`1c.`<*profile*>
`:1`	`2c.`<*profile*>
...	...
`:N-1`	`Nc.`<*profile*> or <*profile*>

In practice, there are hardware-based restrictions for what is permitted to be allocated. So for example on the NVIDIA A100-PCIE-40GB:

`7g.40gb:3` is allowed but

`7g.40gb:4` is not.

Details on supported profiles for hardware can be found in the NVIDIA MIG user guide documentation at `https://docs.nvidia.com/datacenter/tesla/mig-user-guide/supported-mig-profiles.html`, which covers, for example, the A30, A100, H100, and B200 profiles.

Increasing the number of GPU instances can also still be done using the earlier <*number*>`*` prefix syntax together with the colon syntax, if the GPU allows it:

Example

```
[basecm11->device[node001]]% gpusettings; use nvidia:0
[basecm11->device[node001]->gpusettings[nvidia:0]]% set migprofiles 3*2g.10gb:0:0; commit
[basecm11->device[node001]->gpusettings[nvidia:0]]% ..;..
[basecm11->device[node001]]% mig apply
Node             GPU      MIG      Instance Name        Profile  Start    Size
---------------- -------- -------- -------- ----------- -------- -------- --------
node001          0        3        -        2g.10gb     14       0        2
node001          0        3        0        1c.2g.10gb  0        0        1
node001          0        3        1        1c.2g.10gb  0        1        1
node001          0        4        -        2g.10gb     14       2        2
node001          0        4        0        1c.2g.10gb  0        0        1
node001          0        4        1        1c.2g.10gb  0        1        1
node001          0        5        -        2g.10gb     14       4        2
node001          0        5        0        1c.2g.10gb  0        0        1
node001          0        5        1        1c.2g.10gb  0        1        1
```

- **A heterogeneous set of existing profiles** for GPU instances can also be defined for the GPU with MIG profiles.

  For example, 2 instances with the MIG profile `1g.5gb`, and 1 instance with the MIG profile `2g.10gb` can be specified with:

```
[basecm11->device[node001]]% gpusettings; use nvidia:0
[basecm11->device[node001]->gpusettings[nvidia:0]]% set migprofiles 2*1g.5gb 2g.10gb; commit
[basecm11->device[node001]->gpusettings[nvidia:0]]% show
Parameter                      Value
------------------------------ ------------------------------------------------
...
MIG profiles                   2*1g.5gb,2g.10gb
[basecm11->device[node001]->gpusettings[nvidia:0]]% ..;..
[basecm11->device*[node001*]]% mig apply
Node             GPU      MIG      Instance Name     Profile  Start    Size
---------------- -------- -------- -------- -------- -------- -------- --------
node001          0        3        -        2g.10gb  14       0        2
node001          0        3        0        2g.10gb  1        0        2
node001          0        11       -        1g.5gb   19       4        1
node001          0        11       0        1g.5gb   0        0        1
node001          0        13       -        1g.5gb   19       6        1
node001          0        13       0        1g.5gb   0        0        1
```

- **heterogeneous sets are useful when trying to use up all the slices available**, to make the maximum resources available. So, while an administrator can carry out the preceding specification:

  `2*1g.5gb 2g.10gb`

this is not a good allocation of resources since it only makes 4/7 of the GPU slices available, and 4/8 of the memory slices available. An administrator would more sensibly specify something like, for example:

```
5*1g.5gb,2g.10gb
```

which uses up the full 7 GPU slices and 35GB (6/8 slices) of memory available from the physical GPU. This makes full use of the SM resources derived from the physical GPU, so that these SM resources are fully available to workloads.

- **Overallocating slices for MIG configuration is not possible**. If there is an attempt to overallocate, then the slices that are allocated too late are simply not allocated. This can lead to unexpected results for the unwary cluster administrator. For example:
  - `5*1g.5gb,2g.10gb` allocates the 5 slices of the 1g instance and the 2 slices of the 2g instance. But
  - `6*1g.5gb,2g.10gb` allocates the 6 slices of the 1g instance and none of the 2g instance.
  - `7*1g.5gb,2g.10gb` allocates the 7 slices of the 1g instance and none of the 2g instance.
  - `70*1g.5gb,2g.10gb` allocates 7 slices of the 1g instance and none of the 2g instance.

3.18 Configuring Sampling From A Prometheus Exporter

CMDaemon can be configured to sample a Prometheus exporter, for example from the NVIDIA Unified Fabric Manager (UFM) platform. A CMDaemon front end such as `cmsh` can have a data producer, for example `UFM`, configured within the `monitoring setup` mode (section 10.5.4) to allow sampling of the Prometheus exporter:

Example

```
[root@basecm11]# cmsh
[basecm11]% monitoring setup
[basecm11->monitoring->setup]% add prometheus UFM        #create a data producer of type Prometheus
[basecm11->monitoring->setup*[UFM*]]% set urls http://10.180.217.170:9001/metrics     #end point
[basecm11->monitoring->setup*[UFM*]]% set -e NoPostAllowed yes   #HTTP GET only, use for older exporters
[basecm11->monitoring->setup*[UFM*]]% nodeexecutionfilters
[basecm11->monitoring->setup*[UFM*]->nodeexecutionfilters]% active   #run on active head node only
Added active resource filter
[basecm11->monitoring->setup*[UFM*]->nodeexecutionfilters]% commit
```

In the example, the URL needs to be set to the Prometheus export server endpoint. The value of `NoPostAllowed` only needs to be set to `yes` for some older Prometheus versions that do not work with HTTP POST. The data producer is set to run on only the active head node with the `nodeexecutionfilter` setting (page 618).

The preceding example configures CMDaemon to sample from a Prometheus exporter. The other way around, that is to have CMDaemon be the exporter of Prometheus data, can be achieved via the `EnablePrometheusExporterService` directive (page 913).

3.19 Configuring Custom Scripts

Some scripts are used for custom purposes. These are used as replacements for certain default scripts, for example, in the case of non-standard hardware where the default script does not do what is expected. The custom scripts that can be set, along with their associated arguments are:

- `custompowerscript` and `custompowerscriptargument` (section 4.1.4)
- `custompingscript` and `custompingscriptargument` (section 3.19.2)

- customremoteconsolescript and customremoteconsolescriptargument (section 3.19.3)

In addition to the preceding custom* scripts, system information scripts can be set that provide extra information to the sysinfo command in BCM (section 3.19.4).

The environment variables of CMDaemon (section 3.3.1 of the *Developer Manual*) can be used in the scripts. Successful scripts, as is the norm, return 0 on exit.

3.19.1 custompowerscript

The use of custom power scripts is described in section 4.1.4.

3.19.2 custompingscript

The following example script:

Example

```
#!/bin/bash
/bin/ping -c1 $CMD_IP
```

can be defined and set for the cases where the default built-in ping script, cannot be used.

By default, the node device states are detected by the built-in ping script (section 5.5) using ICMP ping. This results in the statuses that can be seen on running the list command of cmsh in device mode. An example output, formatted for convenience, is:

Example

```
[root@basecm11]# cmsh -c "device; format hostname:15, status:15; list"
hostname (key)  status
--------------- ---------------
basecm11        [   UP   ]
node001         [   UP   ]
node002         [   UP   ]
```

If some device is added to the cluster that blocks such pings, then the built-in ping can be replaced by the custom ping of the example, which relies on standard ICMP ping.

However, the replacement custom ping script need not actually use a variety of ping at all. It could be a script running web commands to query a chassis controller, asking if all its devices are up. The script simply has to provide an exit status compatible with expected ping behavior. Thus an exit status of 0 means all the devices are indeed up.

3.19.3 customremoteconsolescript

A custom remote console script can be used to run in the built-in remote console utility. This might be used, for example, to allow the administrator remote console access through a proprietary KVM switch client.

For example, a user may want to run a KVM console access script that is on the head node and with an absolute path on the head node of /root/kvmaccesshack. The script is to run on the console, and intended to be used for node node001, and takes the argument 1. This can then be set in cmsh as follows:

Example

```
[root@basecm11]# cmsh
[basecm11]% device use node001
[basecm11->device[node001]]% get customremoteconsolescript; get customremoteconsolescriptargument

[basecm11->device[node001]]% set customremoteconsolescript /root/kvmaccesshack
[basecm11->device[node001]]% set customremoteconsolescriptargument 1
[basecm11->device[node001]]% rconsole
```

KVM console access session using the 1 *argument option is displayed*

In Base View, the corresponding navigation paths to access these script settings are:

`Devices` > `Nodes` > `Edit` > `Settings` > `Custom remote console script`

and

`Devices` > `Nodes` > `Edit` > `Settings` > `Custom remote console script argument`

while the remote console can be launched via the navigation path:

`Devices` > `Nodes` > `Edit` > `Connect` > `Remote console`

3.19.4 `sysinfo` Custom Scripts

Standard `sysinfo`

The `sysinfo` command in BCM is run from `device` mode in `cmsh` for a node. By default, `sysinfo` returns some basic hardware information for the node, from the OS, from a cache. Direct hardware queries can be carried out with other commands (section 14.11).

Overview Of Running Custom Scripts In `sysinfo`

A cluster administrator may however wish to extract some additional hardware-related information from the cluster. To do this, custom scripts associated with the `sysinfo` command can be created by the cluster administrator. These `sysinfo` custom scripts run when the `sysinfo` command is executed via `cmsh`, and they pick up the additional information.

Custom script types: The scripts can be of three types, with corresponding directory locations. as indicated by the following table:

Type of script	Script directory path on node
local	`/cm/local/apps/cmd/scripts/sysinfo/local/`
director	`/cm/local/apps/cmd/scripts/sysinfo/director/`
head	`/cm/local/apps/cmd/scripts/sysinfo/head/`

Custom script process: When `sysinfo` runs for a particular node, the scripts that are called have the following characteristics, and are run as follows:

- Any scripts of the local type are run on the node that `sysinfo` is executed on. The scripts must be placed by the cluster administrator on the node itself. The node could be a head node, director node, a regular node, or a cloud node. The output from the local scripts is picked up.
- Any scripts of the director type are run on the director node. The scripts must be placed by the cluster administrator on the director node itself. The output from the scripts is picked up.
- Any scripts of the head type are run on the head node. The scripts themselves must be placed by the cluster administrator on the head node. The output from the scripts is picked up.

Custom script output format: The script outputs are JSON format key value pairs (JSON object literals).

The simplest standard JSON output form supported for `sysinfo` is:

```
{
 "key":"value"
}
```

The format if getting output for N key-value pairs is:

```
{
 "key1":"value1",
 "key2":"value2",
      ...
 "keyN":"valueN"
}
```

Nested sysinfo output: The key-value pairs can also be grouped, with the output presented in the following format for a group:

```
{
 "group":{
          "key1":"value1",
          "key2":"value2",
               ...
          "keyN":"valueN"
          }
}
```

The key-value pairs can also be structured with multiple groups. In the following example there are two groups:

Example

```
{
 "group1":{
          "key1.1":"value1.1",
          "key1.2":"value1.2"
          },
 "group2":{
          "key2.1":"value2.1",
          "key2.2":"value2.2",
          "key2.3":"value2.3"
          }
}
```

The nested multilayer output format can be useful for grouped values. For example, `dmidecode` can output key-value pairs that specify starting and ending ranges, which can be grouped according to the various DMI types that are also available in the output.

Simple `sysinfo` custom script construction: For example, the following `bash` script can be run on the head node:

Example

```
[root@basecm11 ~]# cat /cm/local/apps/cmd/scripts/sysinfo/head/outputscript.sh
#!/bin/bash
myhostname=$(hostname)
#next line extracts just the UUID value from the dmidecode output for the system for this particular hardware
myuuid=$(dmidecode | grep -A6 "^System Information" | grep UUID | sed -e 's/^\W*UUID: //')
echo '{'
echo '"script path is":"'$0'",'
echo '"CMDaemon running on":"'$CMD_HOSTNAME'",'
echo '"script running on":"'$myhostname'"',
echo '"'$myhostname' UUID":"'$myuuid'"'
echo '}'
```

If run directly, outside of CMDaemon, then this would display a JSON key-value output similar to the following:

Example

```
[root@basecm11 ~]# /cm/local/apps/cmd/scripts/sysinfo/head/outputscript.sh
{
"script path is":"/cm/local/apps/cmd/scripts/sysinfo/head/outputscript.sh",
"CMDaemon running on":"",
"script running on":"basecm11",
"basecm11 UUID":"6733d33a-2933-41ea-aa3c-b218e784c8b9"
}
```

Custom script placement—overview for placing on a regular node: Placing this head script on a regular node can be done by copying the script into a local directory on the node image, and rebooting the regular node so that it picks up the image with the new local type script. After CMDaemon is updated with the new `sysinfo` information, then, whenever `sysinfo` is run, the script is automatically run by CMDaemon.

Examples Of Running Custom Scripts In `sysinfo`

The following example session makes the preceding concepts more explicit: the script is copied over from the head node to the default image of a regular node. It goes into the directory location for custom `sysinfo` scripts of the local type. Rebooting a node then installs the new script on the node:

Example

```
[root@basecm11 ~]# cp -r /cm/local/apps/cmd/scripts/sysinfo/head/outputscript.sh \
 /cm/images/default-image/cm/local/apps/cmd/scripts/sysinfo/local/
[root@basecm11 ~]# cmsh
[basecm11]% device
[basecm11->device]% reboot node001
     [the reboot of the node has to complete]
```

To update `cmdaemon` with the values from the new scripts for the node, the `--update` option to the `sysinfo` command can be run for the node. Running `sysinfo` for the node then displays the output of the scripts for the node. The `sysinfo` output value for `Age` shows how long it has been since CMDaemon was updated by the scripts for the node:

Example

```
[basecm11->device]% sysinfo node001 --update
[basecm11->device]% sysinfo node001
...

Age                          11s
CMDaemon running on          node001                                                  } local script
script running on            node001                                                  }
node001 UUID                 3b1c7973-07ef-419f-9324-4edf999690d5                     }
script path is               /cm/local/apps/cmd/scripts/sysinfo/local/outputscript.sh }

CMDaemon running on          node001                                                  } head script
script running on            basecm11                                                 }
basecm11 UUID                6733d33a-2933-41ea-aa3c-b218e784c8b9                     }
script path is               /cm/local/apps/cmd/scripts/sysinfo/head/outputscript.sh  }
```

In the preceding example, the grouping braces are not part of the actual output. They are just part of this explanation, and show that the first four lines after the `Age` line are from the local type script running on node001. The four lines after that are from the head type script running on head, even though its corresponding CMDaemon is running on `node001`.

Considerations Before Running Custom Scripts With `sysinfo`

Although BCM gives the administrator the freedom to construct all kinds of custom `sysinfo` scripts, some caution is urged before implementing the scripts. The following issues should at least be considered:

- The scripts should be speedy. The scripts run asynchronously, but a script is expected to take less than 15 seconds to run.
- The data output should be small. JSON objects allow, for example, that megabyte-sized text could be output by the sysinfo scripts. However, this is often unwise, given the nature of clusters. A cluster with a 1000 nodes and 1MB blobs would mean that 1GB of memory is being moved around.

3.20 Cluster Configuration Without Execution By CMDaemon

3.20.1 Cluster Configuration: The Bigger Picture

The configurations carried out in this chapter so far are based almost entirely on configuring nodes, via a CMDaemon front end (`cmsh` or Base View), using CMDaemon to execute the change. Indeed, much of this manual is about this too because it is the preferred technique. It is preferred:

- because it is intended by design to be the easiest way to do common cluster tasks,
- and also generally keeps administration overhead minimal in the long run since it is CMDaemon rather than the system administrator that then takes care of tracking the cluster state.

There are however other cluster configuration techniques besides execution by CMDaemon. To get some perspective on these, it should be noted that cluster configuration techniques are always fundamentally about modifying a cluster so that it functions in a different way. The techniques can then for convenience be separated out into modification techniques that rely on CMDaemon execution and techniques that do not, as follows:

1. **Configuring nodes with execution by CMDaemon:** As explained, this is the preferred technique. The remaining techniques listed here should therefore usually only be considered if the task cannot be done with Base View or `cmsh`.
2. **Replacing the node image:** The image on a node can be replaced by an entirely different one, so that the node can function in another way. This is covered in section 3.20.2. It can be claimed that since it is CMDaemon that selects the image, this technique should perhaps be classed as under item 1. However, since the execution of the change is really carried out by the changed image without CMDaemon running on the image, and because changing the entire image to implement a change of functionality is rather extreme, this technique can be given a special mention outside of CMDaemon execution.
3. **Using a `FrozenFile` directive:** Applied to a configuration file, this directive prevents CMDaemon from executing changes on that file for nodes. During updates, the frozen configuration may therefore need to be changed manually. The prevention of CMDaemon acting on that file prevents the standard cluster functionality that would run based on a fully CMDaemon-controlled cluster configuration. The `FrozenFile` directive is introduced in section 2.6.5, and covered in the configuration context in section 3.20.3.

4. **Using an `initialize` or `finalize` script:** This type of script is run during the initrd stage, much before CMDaemon on the regular node starts up. It is run if the functionality provided by the script is needed before CMDaemon starts up, or if the functionality that is needed cannot be made available later on when CMDaemon is started on the regular nodes. CMDaemon does not execute the functionality of the script itself, but the script is accessed and set on the initrd via a CMDaemon front end (Appendix E.2), and executed during the initrd stage. It is often convenient to carry out minor changes to configuration files inside a specific image in this way, as shown by the example in Appendix E.5. The `initialize` and `finalize` scripts are introduced in section 3.20.4.

5. **A shared directory:** Nodes can be configured to access and execute a particular software stored on a shared directory of the cluster. CMDaemon does not execute the functionality of the software itself, but is able to mount and share directories, as covered in section 3.14.

Finally, outside the stricter scope of cluster configuration adjustment, but nonetheless a broader way to modify how a cluster functions, and therefore mentioned here for more completeness, is:

6. **Software management:** the installation, maintenance, and removal of software packages. Standard post-installation software management based on repositories is covered in sections 9.2–9.6. Third-party software management from outside the repositories, for software that is part of BCM is covered in Chapter 7 of the *Installation Manual*.

   Third-party software that is not part of BCM can be managed on the head node as on any other Linux system, and is often placed under `/opt` or other recommended locations. If required by the other nodes, then the software should typically be set up by the administrator so that it can be accessed via a shared filesystem.

3.20.2 Making Nodes Function Differently By Image

Making All Nodes Function Differently By Image

To change the name of the image used for an entire cluster, for example after cloning the image and modifying it (section 3.20.2), the following methods can be used:

- in Base View, via `Cluster` > `Settings` > `Default software image`
- or in `cmsh` from within the `base` object of `partition` mode

A system administrator more commonly sets the software image on a per-category or per-node basis (section 3.20.2).

Making Some Nodes Function Differently By Image

For minor changes, adjustments can often be made to node settings via initialize and finalize scripts so that nodes or node categories function differently (section 3.20.4).

For major changes on a category of nodes, it is usually more appropriate to have nodes function differently from each other by simply carrying out image changes per node category with CMDaemon. Carrying out image changes per node is also possible. As usual, node settings override category settings.

Modifying images via cloning primitives for a node or category: Setting a changed image for a category can be done as follows with `cmsh`:

1. The image on which the new one will be based is cloned. The cloning operation not only copies all the settings of the original (apart from the name), but also the data of the image:

   Example

```
[root@basecm11 ~]# cmsh
[basecm11]% softwareimage
[basecm11->softwareimage]% clone default-image imagetwo
[basecm11->softwareimage*[imagetwo*]]% commit
... [notice] basecm11: Started to copy: /cm/images/default-image -> /cm/images/imagetwo
[basecm11->softwareimage*[imagetwo*]]%
... [notice] basecm11: Copied: /cm/images/default-image -> /cm/images/imagetwo
[basecm11->softwareimage[imagetwo]]%
```

2. After cloning, the settings can be modified in the new object. For example, if the kernel needs to be changed to suit nodes with different hardware, kernel modules settings are changed (section 5.3.2) and committed. This creates a new image with a new ramdisk.

   Other ways of modifying and committing the image for the nodes are also possible, as discussed in sections 9.2–9.6.

3. The modified image that is to be used by the differently functioning nodes is placed in a new category in order to have the nodes be able to choose the image. To create a new category easily, it can simply be cloned. The image that the category uses is then set:

```
[basecm11->softwareimage[imagetwo]]% category
[basecm11->category]% clone default categorytwo
[basecm11->category*[categorytwo*]]% set softwareimage imagetwo
[basecm11->category*[categorytwo*]]% commit
[basecm11->category[categorytwo]]%
```

4. - For just one node, or a few nodes, the node can be set from `device` mode to the new category (which has the new image):

     ```
     [basecm11->category[categorytwo]]% device
     [basecm11->device]% use node099
     [basecm11->device[node099]]% set category categorytwo
     [basecm11->device*[node099*]]% commit; exit
     ```

   - If there are many nodes, for example node100 sequentially up to node200, they can be set to that category using a `foreach` loop like this:

     Example

     ```
     [basecm11->device]% foreach -n node100..node200 (set category categorytwo)
     [basecm11->device*]% commit
     ```

5. Rebooting restarts the nodes that are assigned to the new category with the new image.

Modifying images by adding files in the `cm/conf` directory, for a category, node, or MAC address: The preceding 5-step method is understandable. For just a few file changes it is perhaps overkill and not very elegant. BCM has a more structured and efficient way to make some nodes function differently by image if only a few file additions are to be carried out. The specification adds the files in the image via a target path that is specified in special configuration locations in the image. It can be configured per node, but also per category and MAC address.

- For a category, the specification takes the form:

  `/cm/images/`<*image*>`/cm/conf/category/`<*category*>`/`<*target*>

  Thus, if some file on the node is to be placed so that on a running node it is at `/path/to/some.file`, and this needs to be configured for an image `default-image`, and a category `default`, then it would be placed at this location on the head node:

Example

```
/cm/images/default-image/cm/conf/category/default/path/to/some.file
```

The file on the target node would be placed in the absolute directory `/path/to/some.file`

Multiple categories can be configured per image. Thus, for example, beside the `default` category, an additional `gpu` category can exist:

Example

```
/cm/images/default-image/cm/conf/category/gpu/path/to/some.file
```

Also, multiple files can be specified per category per image. Thus, beside the file `some.file`, an additional file `some.other.file` could be placed:

Example

```
/cm/images/default-image/cm/conf/category/gpu/path/to/some.file
/cm/images/default-image/cm/conf/category/gpu/path/to/some.other.file
```

- For a node, the configuration form is:

  `/cm/images/`<*image*>`/cm/conf/node/`<*node name*>`/`<*target*>

  An example for a node called `node001` could then be:

  Example

  ```
  /cm/images/default-image/cm/conf/node/node001/path/to/some.file
  ```

- For a MAC address, the configuration form is:

  `/cm/images/`<*image*>`/cm/conf/node/`<*MAC address*>`/`<*target*>

  An example for a node with MAC address `00:aa:bb:cc:dd:ee` could then be:

  Example

  ```
  /cm/images/default-image/cm/conf/node/00-aa-bb-cc-dd-ee/path/to/some.file
  ```

The copying of the specified files to the image is done just before the finalize stage of the node-installer (section 5.4.11) during node provisioning.

A common theme in BCM is that node-level configuration overrides category-level configuration. In keeping with this behavior, a file configuration at category level could be applied to the many nodes in a category. And, a file configuration copy at node level (for a node that is in the category) overrides the category level value for just that particular node.

3.20.3 Making All Nodes Function Differently From Normal Cluster Behavior With `FrozenFile`

Configuration changes carried out by Base View or `cmsh` often generate, restore, or modify configuration files (Appendix A).

However, sometimes an administrator may need to make a direct change (without using Base View or `cmsh`) to a configuration file to set up a special configuration that cannot otherwise be done.

The `FrozenFile` directive to CMDaemon (Appendix C, page 920) applied to such a configuration file stops CMDaemon from altering the file. The frozen configuration file is generally applicable to all nodes and is therefore a possible way of making all nodes function differently from their standard behavior.

Freezing files is however best avoided, if possible, in favor of a CMDaemon-based method of configuring nodes, for the sake of administrative maintainability.

3.20.4 Adding Functionality To Nodes Via An `initialize` Or `finalize` Script

CMDaemon can normally be used to allocate different images per node or node category as explained in section 3.20.2. However, some configuration files do not survive a reboot (Appendix A), sometimes hardware issues can prevent a consistent end configuration, and sometimes drivers need to be initialized before provisioning of an image can happen. In such cases, an `initialize` or `finalize` script (sections 5.4.5, 5.4.11, and Appendix E.5) can be used to initialize or configure nodes or node categories.

These scripts are also useful because they can be used to implement minor changes across nodes:

Example

Supposing that some nodes with a particular network interface have a problem auto-negotiating their network speed, and default to 100Mbps instead of the maximum speed of 1000Mbps. Such nodes can be set to ignore auto-negotiation and be forced to use the 1000Mbps speed by using the `ETHTOOL_OPTS` configuration parameter in their network interface configuration file: `/etc/sysconfig/network-scripts/ifcfg-eth0` (or `/etc/sysconfig/network/ifcfg-eth0` in SUSE).

The `ETHTOOL_OPTS` parameter takes the options to the "`ethtool -s` *<device>*" command as options. The value of *<device>* (for example `eth0`) is specified by the filename that is used by the configuration file itself (for example `/etc/sysconfig/network-scripts/ifcfg-eth0`). The `ethtool` package is installed by default with BCM. Running the command:

```
ethtool -s autoneg off speed 1000 duplex full
```

turns out after some testing to be enough to reliably get the network card up and running at 1000Mbps on the problem hardware.

However, since the network configuration file is overwritten by node-installer settings during reboot, a way to bring persistence to the file setting is needed. One way to ensure persistence is to append the configuration setting to the file with a `finalize` script, so that it gets tagged onto the end of the configuration setting that the node-installer places for the file, just before the network interfaces are taken down again in preparation for `init`.

The script may thus look something like this for a Red Hat system:

```
#!/bin/bash

## node010..node014 get forced to 1000 duplex
if [[ $CMD_HOSTNAME = node01[0-4] ]]
then
echo 'ETHTOOL_OPTS="speed 1000 duplex full"'>>/localdisk/etc/sysconfig/network-scripts/ifcfg-eth0
fi
```

The method of enforcing an interface space just outlined is actually just for educational illustration, and is not a recommended method.

In practice, the recommended way to enforce an interface speed is to simply set it in the CMDaemon database. For example, for the boot interface of `node001` it could be via the Base View navigation path:

`Devices` > `Nodes[node001]` > `Edit` > `Settings` > `Interfaces[BOOTIF]` > `Edit` > `Speed`

3.20.5 Examples Of Configuring Nodes With Or Without CMDaemon

A node or node category can often have its software configured in CMDaemon via Base View or `cmsh`:

Example

Configuring a software for nodes using Base View or `cmsh`: If the software under consideration is CUPS, then a node or node category can manage it from Base View or `cmsh` as outlined in section 3.15.2.

A counterexample to this is:

Example

Configuring a software for nodes without using Base View or cmsh, using an image: Software images can be created with and without CUPS configured, without using Base View or cmsh except to link nodes to their appropriate image via the associated category. Setting up nodes to load one of these two images via a node category is an alternative way of letting nodes run CUPS.

Whether node configuration for a particular functionality is done with CMDaemon, or directly with the software, depends on what an administrator prefers. In the preceding two examples, the first example, that is the one with Base View or cmsh setting the CUPS service, is likely to be preferred over the second example, where an entire separate image must be maintained. A new category must also be created in the second case.

Generally, sometimes configuring the node via BCM, and not having to manage images is better, sometimes configuring the software and making various images to be managed out of it is better, and sometimes only one of these techniques is possible anyway.

Configuring Nodes Using Base View Or cmsh: Category Settings

When configuring nodes using Base View or cmsh, configuring particular nodes from a node category to overrule the state of the rest of its category (as explained in section 2.1.3) is sensible for a small number of nodes. For larger numbers it may not be organizationally practical to do this, and another category can instead be created to handle nodes with the changes conveniently.

The CUPS service in the next two examples is carried out by implementing the changes via Base View or cmsh acting on CMDaemon.

Example

Setting a few nodes in a category: If only a few nodes in a category are to run CUPS, then it can be done by enabling CUPs just for those few nodes, thereby overriding (section 2.1.3) the category settings.

Example

Setting many nodes to a category: If there are many nodes that are to be set to run CUPS, then a separate, new category can be created (cloning it from the existing one is easiest) and those many nodes are moved into that category, while the image is kept unchanged. The CUPS service setting is then set at category level to the appropriate value for the new category.

In contrast to these two examples, the software image method used in section 3.20.2 to implement a functionality such as CUPS would load up CUPS as configured in an image, and would not handle it via CMDaemon (except to link nodes to their appropriate image via the associated category). So, in section 3.20.2, software images prepared by the administrator are set for a node category. Since, by design, images are only selected for a category, a node cannot override the image used by the category other than by creating a new category, and using it with the new image. The administrative overhead of this can be inconvenient.

Administrators would therefore normally prefer letting CMDaemon track software functionality across nodes as in the last two examples, rather than having to deal with tracking software images manually. Indeed, the roles assignment option (section 2.1.5) is just a special pre-configured functionality toggle that allows CMDaemon to set categories or regular nodes to provide certain functions, typically by enabling services.

3.21 Saving A Backup Of Configuration Files With versionconfigfiles

If versionconfigfiles is set to the value yes for a node or a category, then if configuration files changed for that node or category due to CMDaemon, then the old configuration files are saved.

Example

```
[root@basecm11 ~]# cmsh
[basecm11]% device use node001
[basecm11->device[node001]]% set versionconfigfiles yes; commit
```

This is useful, for example, if an administrator would like to know what the configuration was just before it was changed.

If a configuration change takes place, then the old configuration files are automatically sent from the node where they changed, to the active head node. The configuration files:

- are saved on the active head node under the directory `/var/spool/cmd/config_file_versions`, under their node name.
- have a modification time that indicates the time of the change.
- are given a suffix in the form of the local unix epoch time.

Example

```
[root@basecm11 ~]# cd /var/spool/cmd/
[root@basecm11 cmd]# tree -a --charset=C config_file_versions/
config_file_versions/
|-- node001
|   |-- cm
|   |   `-- local
|   |       `-- modulefiles
|   |           `-- slurm
|   |               |-- .modulerc.lua.1970-01-01_01:00:00
|   |               `-- slurm
|   |                   `-- 21.08.8.1970-01-01_01:00:00
...

[root@basecm11 cmd]# cd config_file_versions/node001/cm/local/modulefiles/slurm/
[root@basecm11 slurm]# ls -al .modulerc.lua.1970-01-01_01:00:00
-rw-r--r-- 1 root root 43 Mar 14 17:22 .modulerc.lua.1970-01-01_01:00:00
[root@basecm11 slurm]#
```

4

Power Management

Aspects of power management in NVIDIA Base Command Manager include:

- managing the main power supply to nodes through the use of power distribution units, baseboard management controllers, or CMDaemon
- monitoring power consumption over time
- setting power-saving options in workload managers
- ensuring the passive head node can safely take over from the active head during failover (Chapter 15)
- allowing cluster burn tests to be carried out (Chapter 11 of the *Installation Manual*)

The ability to control power inside a cluster is therefore important for cluster administration, and also creates opportunities for power savings. This chapter describes BCM power management features.

In section 4.1 the configuration of the methods used for power operations is described.

Section 4.2 then describes the way the power operations commands themselves are used to allow the administrator turn power on or off, reset the power, and retrieve the power status. It explains how these operations can be applied to devices in various ways.

Section 4.3 describes power infrastructure overview commands. This is typically managed at rack level and at data center level, and is covered in more detail in chapters 3 and 4 of the *NVIDIA Mission Control Manual*.

Section 4.4 briefly covers monitoring power.

Section 4.5 is a note about switch configuration to gracefully handle power downs of devices on the network.

The integration of power saving with workload management systems is covered in the chapter on Workload Management (section 7.9).

4.1 Configuring Power Parameters

Several methods exist to control power to devices:

- Power Distribution Unit (PDU) based power control
- IPMI-based power control (for node devices only)
- Custom power control
- HP iLO-based power control (for node devices only)
- Dell DRAC-based power control (for node devices only)
- Cisco UCS CIMC-based power control (for node devices only)
- Redfish-based power control (for node devices only)

4.1.1 PDU-based Power Control

Introduction To PDU-based Power Control

For PDU-based power control, the power supply of a device is plugged into a port on a PDU. The device can be a node, but also anything else with a power supply, such as a switch or a blade chassis. One PDU can even supply power to another PDU, although that typically implies a suboptimal power supply layout if the supplies are comparable in size.

A device that is supplied by a PDU can then be turned on or off by changing the state of the PDU port.

Configuring The PDU Itself

To use PDU-based power control, the PDU itself must be added and configured as a device in the cluster, and must be reachable over the network. PDU configuration was introduced in section 3.8.

- The PDU can be added using `cmsh` using `device` mode, and set as an object with a `type` of `PowerDeviceUnit`, with a name of `mypdu`. using:

  Example

  ```
  root@basecm11:~# cmsh
  [basecm11]% device add powerdistributionunit mypdu
  [basecm11->device*[mypdu*]]% commit
  ```

  Values can be viewed with the `show` command, and set with the `set` command.

- The PDU can be added using Base View with the navigation path:

  ```
  Devices > All Devices > + Add > PowerDistributionUnit
  ```

  Values can be set for its properties in the pane that opens up.

Configuring The Devices To Use The PDU

After the PDU itself is configured, then the devices that use it can be configured to use the PDU and its ports.

A node `node001` that is to be powered by the PDU `mypdu` using its power outlet port 1, can be configured as follows:

- Using `cmsh` with:

  ```
  [basecm11->device[node001]]% set powerdistributionunits mypdu:1
  [basecm11->device*[node001*]]% commit
  [basecm11->device[node001]]%
  ```

- Using Base View with the navigation path:

  ```
  Devices > All Devices > node001 > Settings > Power distribution units > +
  ADD
  ```

  which allows the PDU and its port (the `PDUPort` components) to be selected (figure 4.1):

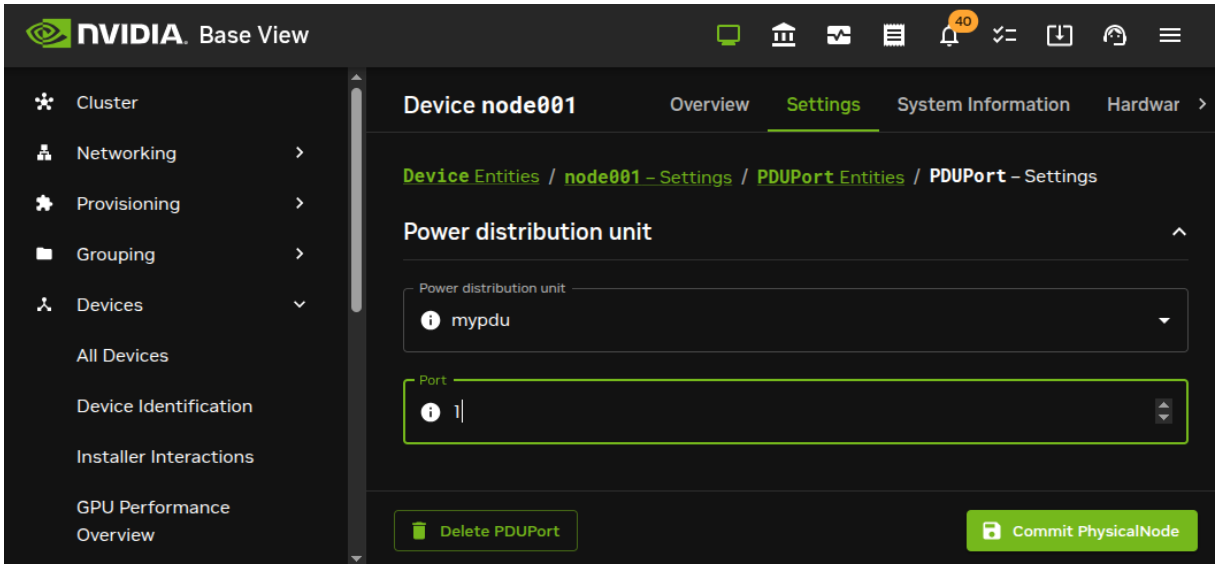


Figure 4.1: PDU configuration settings for a node

For the APC brand of PDUs: the `Power control` property (page 147) should be set to `apc`, or the list of PDU ports is ignored by default. Overriding the default is described in section 4.1.3.

Power Ports: One-to-many And Many-to-one

Nodes may have multiple power feeds for redundancy reasons. Thus, there may be multiple PDU ports and multiple PDUs defined for a single device. The cluster management takes care of operating all ports of a device in the correct order when a power operation is done on the device.

For example, if a PDU `mypdu` has its ports 2 and 4 connected to a blade chassis `mychassis`, then the configuration can be specified using `cmsh` with:

Example

```
[basecm11->device*[mychassis]]% set powerdistributionunits mypdu:2 mypdu:4; commit
```

It is also possible for multiple devices to share the same PDU port. This is the case for example when *twin nodes* are used (i.e. two nodes sharing a single power supply). In this case, all power operations on one device apply to all nodes sharing the same PDU port.

Non-manageable PDUs

If the PDUs defined for a node are not manageable, then the node's baseboard management controllers (that is, IPMI/iLO and similar) are assumed to be inoperative and are therefore assigned an unknown state. This means that dumb PDUs, which cannot be managed remotely, are best not assigned to nodes in BCM. It is suggested that administrators record that a dumb PDU is assigned to a node as follows:

- in Base View the `Notes` field or the `Userdefined1`/`Userdefined2` fields can be used with the navigation paths:

  `Devices > All Devices[`*node*`] > Settings > Partition > Notes`
  or
  `Devices > All Devices[`*node*`] > Settings > User Defined > Userdefined1/Userdefined1`

- in `cmsh` the equivalent is accessible on using the device from `device` mode, and running:

- set notes
- set userdefined1 or
- set userdefined2

Manageable PDUs And Node Power Status

For PDUs that are manageable:

- In cmsh, power-related options can be accessed from device mode, after selecting a device:

Example

```
[basecm11]% device use node001
[basecm11->device[node001]]% show | grep -i power
Custom power script argument
Ipmi/iLO power reset delay             0
Power control                          apc
PowerDistributionUnits                 apc01:6 apc01:7
```

The power status of a node can be accessed with:

Example

```
[basecm11->device[node001]]% power status
```

NOTE: The output of power status is delivered as a **private** event (section 10.10), unlike device status/status, which is not event-based. If private events have been turned off with events off (or events off private), then power status produces no output at all. This is because power queries can be slow, involving a round trip through PDUs or BMCs, while device status is fast and does not need the event indirection. Running events off broadcast instead, leaving private events on, still allows power status output to be seen.

If the node is up and has one or more PDUs assigned to it, then the power status is one of ON, OFF, RESET, FAILED, or UNKNOWN:

Node Power Status	Description
ON	Power is on
OFF	Power is off
RESET	Shows during the short time the power is off during a power reset. The reset is a hard power off for PDUs, but can be a soft or hard reset for other power control devices.
FAILED	Power status script communication failure.
UNKNOWN	Power status script timeout

Rack power status: For a rack in a DGX system, the power status command can show some other values:

Rack Power Status	Description
DONE	Power on completed
GOOD	Check before power status is checked is fine
BUSY	Waiting for power status response
FAIL	Power status check failure
PENDING	Waiting to run power status
CANCELLED	Power on cancelled

4.1.2 IPMI-Based Power Control

IPMI-based power control relies on the baseboard management controller (BMC) inside a node. It is therefore only available for node devices. Blades inside a blade chassis typically use IPMI for power management. Section 3.7 describes setting up networking and authentication for IPMI/iLO/DRAC/CIMC/Redfish interfaces.

To carry out IPMI-based power control operations, the `Power control` property (page 147) must be set to the IPMI interface through which power operations should be relayed. Normally this IPMI interface is configured to be `ipmi0`. Any list of configured APC PDU ports displayed in the GUI is ignored by default when the `Power control` property is not `apc`.

Example

Configuring power parameters settings for all the nodes using `cmsh`, with IPMI interfaces that are called `ipmi0`:

```
[mycluster]% device
[...device]% foreach -t physicalnode (set powercontrol ipmi0; commit)
```

Example

Configuring power parameters settings for a node using `cmsh` with APC:

```
[mycluster]% device use node001
[...device[node001]]% set powerdistributionunits apc01:6 apc01:7 apc01:8
[...device*[node001*]]% get powerdistributionunits
apc01:6 apc01:7 apc01:8
[...device*[node001*]]% removefrom powerdistributionunits apc01:7
[...device*[node001*]]% get powerdistributionunits
apc01:6 apc01:8
[...device*[node001*]]% set powercontrol apc
[...device*[node001*]]% get powercontrol
apc
[...device*[node001*]]% commit
```

4.1.3 Combining PDU- and IPMI-Based Power Control

By default when nodes are configured for IPMI Based Power Control, any configured PDU ports are ignored. However, it is sometimes useful to change this behavior.

For example, in the CMDaemon configuration file directives in `/cm/local/apps/cmd/etc/cmd.conf` (introduced in section 2.6.2 and listed in Appendix C), the default value of `PowerOffPDUOutlet` is `false`. It can be set to `true` on the head node, and CMDaemon restarted to activate it.

With `PowerOffPDUOutlet` set to `true` it means that CMDaemon, after receiving an IPMI-based power off instruction for a node, and after powering off that node, also subsequently powers off the PDU port. Powering off the PDU port shuts down the BMC, which saves some additional power—typically a few watts per node. When multiple nodes share the same PDU port, the PDU port only powers off when all nodes served by that particular PDU port are powered off.

When a node has to be started up again the power is restored to the node. It is important that the node BIOS is configured to automatically power on the node when power is restored.

4.1.4 Custom Power Control

For a device which cannot be controlled through any of the standard existing power control options, it is possible to set a custom power management script. This is then invoked by the cluster management daemon on the head node whenever a power operation for the device is done.

Power operations are described further in section 4.2.

Using `custompowerscript`

To set a custom power management script for a device, the `powercontrol` attribute is set by the administrator to `custom` using either Base View or `cmsh`, and the value of `custompowerscript` is specified by the administrator. The value for `custompowerscript` is the full path to an executable custom power management script on the head node(s) of a cluster.

A custom power script is invoked with the following mandatory arguments:

```
myscript <operation> <device>
```

where `<device>` is the name of the device on which the power operation is done, and `<operation>` is one of the following:

```
ON
OFF
RESET
STATUS
```

On success a custom power script exits with exit code 0. On failure, the script exits with a non-zero exit-code.

Using `custompowerscriptargument`

The mandatory argument values for `<operation>` and `<device>` are passed to a custom script for processing. For example, in `bash` the positional variables `$1` and `$2` are typically used for a custom power script. A custom power script can also be passed a further argument value by setting the value of `custompowerscriptargument` for the node via `cmsh` or Base View. This further argument value would then be passed to the positional variable `$3` in `bash`.

An example custom power script is located at `/cm/local/examples/cmd/custompower`. In it, setting `$3` to a positive integer delays the script via a `sleep` command by `$3` seconds.

An example that is conceivably more useful than a "`sleep $3`" command is to have a "`wakeonlan $3`" command instead. If the `custompowerscriptargument` value is set to the MAC address of the node, that means the MAC value is passed on to `$3`. Using this technique, the power operation ON can then carry out a Wake On LAN operation on the node from the head node.

Setting the `custompowerscriptargument` can be done like this for all nodes:

```
#!/bin/bash
for nodename in $(cmsh -c "device; foreach * (get hostname)")
do
  macad=`cmsh -c "device use $nodename; get mac"`
  cmsh -c "device use $nodename; set customscriptargument $macad; commit"
done
```

The preceding material usefully illustrates how `custompowerscriptargument` can be used to pass on arbitrary parameters for execution to a custom script.

However, the goal of the task can be achieved in a simpler and quicker way using the environment variables available in the cluster management daemon environment (section 3.3.1 of the *Developer Manual*). This is explained next.

Using Environment Variables With custompowerscript

Simplification of the steps needed for custom scripts in CMDaemon is often possible because there are values in the CMDaemon environment already available to the script. A line such as:

```
env > /tmp/env
```

added to the start of a custom script dumps the names and values of the environment variables to /tmp/env for viewing.

One of the names is $CMD_MAC, and it holds the MAC address string of the node being considered.

So, it is not necessary to retrieve a MAC value for custompowerscriptargument with a bash script as shown in the previous section, and then pass the argument via $3 such as done in the command "wakeonlan $3". Instead, custompowerscript can simply call "wakeonlan $CMD_MAC" directly in the script when run as a power operation command from within CMDaemon.

4.1.5 Hewlett Packard iLO-Based Power Control

iLO Configuration During Installation

If "Hewlett Packard" is chosen as the node manufacturer during installation (section 3.3.11 of the *Installation Manual*), and the nodes have an iLO management interface, then Hewlett-Packard's iLO management package, hponcfg, is installed by default on the nodes and head nodes.

iLO Configuration After Installation

If "Hewlett Packard" has not been specified as the node manufacturer during installation then it can be configured after installation as follows:

The hponcfg rpm package is normally obtained and upgraded for specific HP hardware from the HP website. Using an example of hponcfg-3.1.1-0.noarch.rpm as the package downloaded from the HP website, and to be installed, the installation can then be done on the head node, the software image, and in the node-installer as follows:

```
rpm -iv hponcfg-3.1.1-0.noarch.rpm
rpm --root /cm/images/default-image -iv hponcfg-3.1.1-0.noarch.rpm
rpm --root /cm/node-installer -iv hponcfg-3.1.1-0.noarch.rpm
```

To use iLO on a node, the iLO interface of the node is set up just like the IPMI interfaces as outlined in section 4.1.2. That is, using "set powercontrol ilo0" instead of "set powercontrol ipmi0". BCM treats HP iLO interfaces just like regular IPMI interfaces, except that the interface names are ilo0, ilo1... instead of ipmi0, ipmi1...

For example, nodes in the default category can be brought under iLO power control as follows:

Example

```
[mycluster]% device foreach -c default (set powercontrol ilo0)
[mycluster]% device commit
```

4.1.6 Dell drac-based Power Control

Dell drac configuration is covered on page 142.

4.1.7 Redfish-Based and CIMC-Based Power Control

Section 3.7 describes setting up networking and authentication for Redfish/CIMC, as well as for IPMI/iLO/DRAC interfaces.

4.2 Power Operations

4.2.1 Power Operations Overview

Main Power Operations

Power operations may be carried out on devices from either Base View or cmsh. There are four main power operations:

- `power on`
- `power off`
- `power reset`: power off a device and power it on again after a brief delay (is not implemented for racks)
- `power status`: check the power status and display a response

Scheduling-related Power Operations

There are also *scheduling-related* power operations, which are currently (December 2018) only accessible via `cmsh`. Scheduling-related power operations are power operations associated with managing and viewing explicitly-scheduled execution.

Scheduled execution of power operations can be carried out explicitly via the `--at`, `--after`, `-d`, and `--parallel-delay` options. The scheduling-related power operations to manage and view such scheduled power operations are:

- `power wait`: Identifies the devices that have power operations that are in the `waiting` state, i.e. waiting to be carried out, and also outputs the number of operations that are waiting to be carried out.
- `power cancel`: Cancels an operation in the `waiting` state (or `busy` state for racks). The devices on which they should be cancelled can be specified.
- `power list`: Lists the power operations on the device and the states of the operations. Possible states for the operations are:
  - `waiting`: waiting to be executed
  - `busy`: are being executed
  - `cancelled`: have been cancelled
  - `done`: have been executed

It is possible that power operations without an explicitly-scheduled execution time setting show up very briefly in the output of `power list` and `power wait`. However, the output displayed is almost always about the explicitly-scheduled power operations.

4.2.2 Power Operations With Base View

In Base View, executing the main power operations can be carried out as follows:

- using the vertical dots option within the `Actions` column for a device. This can be accessed for a device—for example the head node—using the navigation path:

  `Devices > All Devices > Head Node >` ⋮

  Clicking on the vertical dots then makes power operation menu items available (figure 4.2):

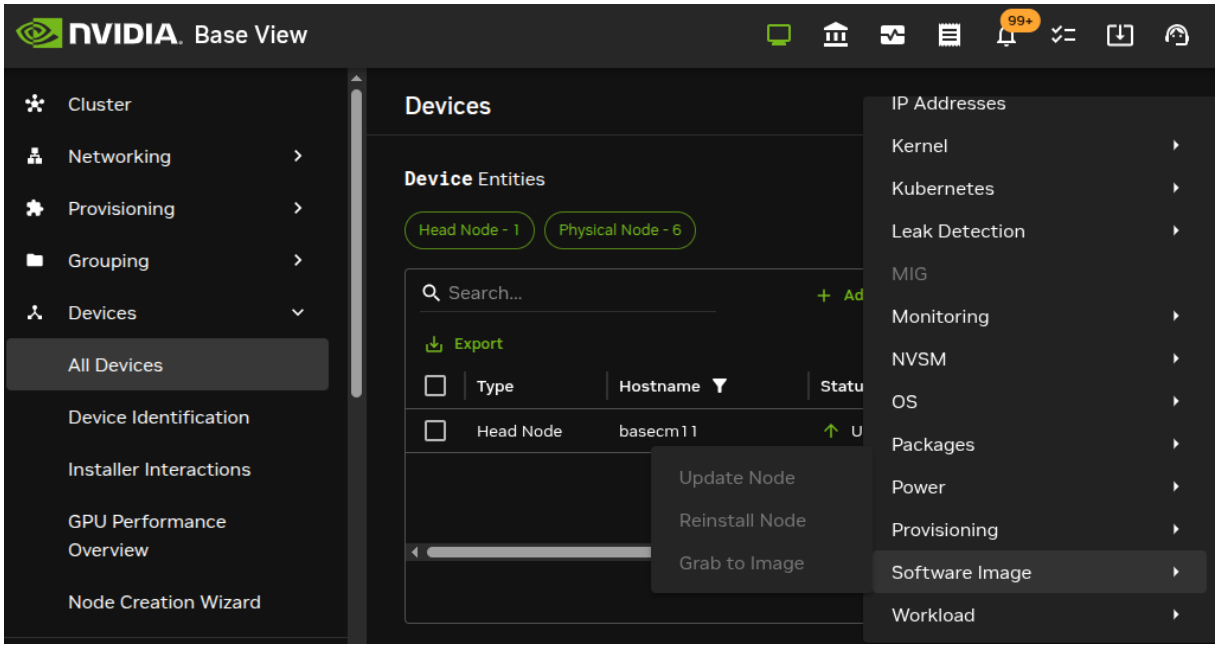


Figure 4.2: `Actions` button, accessing the power operations

- using the vertical dots option within the `Actions` column for categories or nodegroups. This can be accessed for a category—for example the `default` category—using the navigation path:

  `Grouping > Categories > default >` ⋮

4.2.3 Power Operations Through `cmsh`

Power operations on nodes can be carried out from within the `device` mode of `cmsh`, via the `power` command options.

Powering On

Powering on can be carried out on a list of nodes (page 75). Powering on `node001`, and nodes from `node018` to `node033` (output truncated):

Example

```
[mycluster]% device power -n node001,node018..node033 on
apc01:1 ............. [   ON    ] node001
apc02:8 ............. [   ON    ] node018
apc02:9 ............. [   ON    ] node019
...
```

When a power operation is carried out on multiple devices, CMDaemon ensures that a 1 second delay occurs by default between successive devices. This helps avoid power surges on the infrastructure.

Delay Period Between Nodes

The delay period can be modified from within the `device` mode of `cmsh`, by using the `-d|--delay` option of the `power` command. For example, the preceding power command can be run with a shorter, 10ms delay with:

```
[mycluster]% device power -n node001,node018..node033 -d 0.01 on
```

A 0-second delay (`-d 0`) should not be set for larger number of nodes, unless the power surge that this would cause has been taken into consideration.

Powering Up In Batches

Groups of nodes can be powered up "in batches", according to power surge considerations. For example, to power up 3 racks at a time ("in batches of 3"), the `-p|--parallel` option is used:

Example

```
[mycluster]% device power on -p 3 rack[01-12]
```

By default, there is a delay of 20s between batch commands. So, in the preceding example, there is a 20s pause before the each batch of the next three racks is powered up. For batch operation a delay of `-d 0` is assumed, i.e. the nodes within in the rack are powered up without a built-in delay between the nodes of the rack.

Thread Use During Powering Up

The default number of threads that are started up to handle powering up of all the nodes is 32. If the hardware can cope with it, then it is possible to decrease startup time by increasing the default number of threads used to handle powerup, by editing the `PowerThreadPoolSize` advanced configuration directive in CMDaemon (page 919).

Powering Off Nodes

An example of powering off nodes is the following, where all nodes in the `default` category are powered off, with a 100ms delay between nodes (some output elided):

Example

```
[mycluster]% device power off -c default -d 0.1
apc01:1 ............. [   OFF   ] node001
apc01:2 ............. [   OFF   ] node002
...
apc23:8 ............. [   OFF   ] node953
```

Getting The Power Status

The `power status` command lists the status for devices:

Example

```
[mycluster]% device power status -g mygroup
apc01:3 ............. [   ON    ] node003
apc01:4 ............. [   OFF   ] node004
```

Getting The Power History

The `power history` command lists the last few power operations on nodes. By default it lists up to the last 8.

Example

```
[mycluster]% device power history
Device   Time                     Operation    Success
-------- ------------------------ ------------ ------------
node001  Sat Sep 14 03:35:03 2019 shutdown     yes
node001  Fri Sep 20 14:28:38 2019 on           yes
node002  Sat Sep 14 03:35:03 2019 shutdown     yes
node002  Fri Sep 20 14:28:38 2019 on           yes
node003  Sat Sep 14 03:35:03 2019 shutdown     yes
node003  Fri Sep 20 14:28:38 2019 on           yes
node004  Sat Sep 14 03:35:03 2019 shutdown     yes
```

The power Command Help Text

The help text for the power command in device mode is:

```
[basecm11->device]% help power
Name:
          power - Manipulate or retrieve power state of devices

Usage:

          power [OPTIONS] status
          power [OPTIONS] on
          power [OPTIONS] off
          power [OPTIONS] reset
          power [OPTIONS] list
          power [OPTIONS] history
          power [OPTIONS] cancel
          power [OPTIONS] wait <index>

Options:
          -n, --nodes <node>
               List of nodes, e.g. node001..node015,node020..node028,node030 or
               ^/some/file/containing/hostnames

          -g, --group <group>
               Include all nodes that belong to the node group, e.g. testnodes or test01,test03

          -c, --category <category>
               Include all nodes that belong to the category, e.g. default or default,gpu

          -r, --rack <rack>
               Include all nodes that are located in the given rack, e.g rack01 or
               rack01..rack04

          -h, --chassis <chassis>
               Include all nodes that are located in the given chassis, e.g chassis01 or
               chassis03..chassis05

          -e, --overlay <overlay>
               Include all nodes that are part of the given overlay, e.g overlay1 or
               overlayA,overlayC

          -m, --image <image>
               Include all nodes that have the given image, e.g default-image or
               default-image,gpu-image

          -t, --type <type>
               Type of devices, e.g node or virtualnode,cloudnode

          -i, --intersection
               Calculate the intersection of the above selections

          -u, --union
               Calculate the union of the above selections

          -l, --role role
               Filter all nodes that have the given role
```

```
        -s, --status <status>
             Only run command on nodes with specified status, e.g. UP, "CLOSED|DOWN",
             "INST.*"

        -b, --background
             Run in background, output will come as events

        -d, --delay <seconds>
             Wait <seconds> between executing two sequential power commands. This option is
             ignored for the status command

        -f, --force
             Force power command on devices which have been closed

        -w, --overview
             Group all power operation results into an overview

        -p, --parallel <number>
             Number of parallel option-items to be used per batch, default 0 (disabled)

        --at <time>
             Execute the operation at the provided time

        --after <seconds>
             Wait <seconds> before executing the operation

        --parallel-delay <seconds>
             Wait <seconds> between executing the next batch of parallel commands, default
             20s

        --parallel-dry-run
             Only display the times at which operations will be executed, do not perform
             any power operations

        --retry-count <number>
             Number of times to retry operation if it failed the first time (default 0)

        --retry-delay <seconds>
             Delay between consecutive tries of a failed power operation (default 3s)

        --port <pdu>:<port>
             Do the power operation directly on a pdu port.

Examples:
        power status               Display power status for all devices or current device
        power on node001           Power on node001
        power on -n node00[1-2]    Power on node001 and node002
        power list                 List all pending power operations
        power history              List the last couple of power operations
        power wait                 List all power operation that can be waited for
        power wait 1               Wait for a power operation to be completed
        power wait all             Wait for all power operations to be completed
        power wait last            Wait for the last given power operation to be completed
        power off --after 10m      Power off the current node after 10 minutes
```

```
power off --at 23:55       Power off the current node today just before midnight
power cancel node001       Cancel all pending power operations for node001
power on -p 4 rack[01-80]  Power on racks 1 to 80 in batches of 4. With a delay of 20s
                           between each batch. And a delay of 0s between nodes.
power on --port pdu1:1     Power on port 1 on pdu1
power on --port pdu1:[1-4] Power on port 1 through 4 on pdu1
```

4.3 Power Infrastructure Overview

The `device` mode of `cmsh` provides several overview commands for inspecting the state of power infrastructure components.

4.3.1 PDU Overview

The `pduoverview` command displays information about Power Distribution Units (PDUs) and their connected devices:

Example

```
[basecm11->device]% powerdistributionunit
Device  : A02-P1-PDU-01
State   : [  UP  ]
Model   :
Banks   : 2
Outlets : 24
...
```

4.3.2 CDU Overview

The `cduoverview` command displays an overview of Cooling Distribution Units (CDUs) for liquid-cooled systems. It shows supply and return temperatures, differential and system pressure, liquid flow rate, and the racks served by each CDU:

Example

```
[basecm11->device]% cduoverview
Hostname  Supply temp  Return temp  Diff pressure  Sys pressure  Flow   Available  Status  Racks
--------- ------------ ------------ -------------- ------------- ------ ---------- ------- ------
cdu01     18.5         23.1         0.42           2.1           12.3   yes        ok      A05
...
```

CDU data is provided by a conforming Building Management System (BMS) using MQTT. The integration of BMS with BCM is covered in section 3.3.1 of the *NVIDIA Mission Control Manual*.

An example list of CDU measurables is given in section section 12.5 of the *NVIDIA Mission Control Manual*.

4.3.3 Power Shelf Overview

Power shelf configuration is covered in Chapter 4 of the *NVIDIA Mission Control Manual*.

The `powershelfoverview` command (section 4.2 of the *NVIDIA Mission Control Manual*) displays an overview of a particular power shelf, showing the measurements from its PSUs (6 in the following example):

Example

```
[basecm11->device*[B05-P1-PWR-01]]% powershelfoverview
        Input  Input    Input     Rail   Rail     Rail                  Fan
Supply  power  current  voltage   power  current  voltage  Temperature  speed      Healthy
```

```
------- ------ -------- --------- ------ -------- -------- ------------ ---------  --------
1       679 W  2.98 A   240.75 V  652 W  13.06 A  49.85 V  44 C         6.7 KRPM   PASS
2       632 W  2.8 A    240.75 V  607 W  12.18 A  49.87 V  44 C         6.6 KRPM   PASS
3       599 W  2.64 A   240.75 V  575 W  11.48 A  49.87 V  43 C         6.8 KRPM   PASS
4       599 W  2.63 A   240.75 V  575 W  11.51 A  49.89 V  43 C         6.8 KRPM   PASS
5       617 W  2.76 A   240.5 V   574 W  11.5 A   49.87 V  43 C         6.9 KRPM   PASS
6       609 W  2.64 A   240.5 V   584 W  11.7 A   49.87 V  43 C         6.6 KRPM   PASS
```

An example list of power shelf measurables is given in section 12.4 of the *NVIDIA Mission Control Manual*.

4.4 Monitoring Power

Monitoring power consumption is important since electrical power is an important component of the total cost of ownership for a cluster. The monitoring system of BCM collects power-related data from PDUs in the following metrics:

- `PDUBankLoad`: Phase load (in amperes) for one (specified) bank in a PDU
- `PDULoad`: Total phase load (in amperes) for one PDU

Chapter 10 on cluster monitoring has more on metrics and how they can be visualized.

4.5 Switch Configuration To Survive Other Device Power Downs

Besides the nodes and the BMC interfaces being configured for power control, it may be necessary to check that switches can handle power on and off network operations properly. Interfaces typically negotiate the link speed down to reduce power while still supporting Wake On Lan and other features. During such renegotiations the switch may lose connectivity to the node or BMC interface. This can happen if dynamic speed negotiation is disabled on the switch. Dynamic speed negotiation should therefore be configured to be on on the switch in order to reduce the chance that the state of devices on the networks is not misunderstood, so that a node does not, for example, try to provision from a node in a powered-down state,

5

Node Provisioning

This chapter covers *node provisioning*. Node provisioning is the process of how nodes obtain an image. Typically, this happens during their stages of progress from power-up to becoming active in a cluster, but node provisioning can also take place when updating a running node.

Section 5.1 describes the stages leading up to the loading of the kernel onto the node.

Section 5.2 covers configuration and behavior of the provisioning nodes that supply the software images.

Section 5.3 describes the configuration and loading of the kernel, the ramdisk, and kernel modules.

Section 5.4 elaborates on how the node-installer identifies and places the software image on the node in a 13-step process.

Section 5.5 explains node states during normal boot, as well node states that indicate boot problems.

Section 5.6 describes how running nodes can be updated, and modifications that can be done to the update process.

Section 5.7 explains how to add new nodes to a cluster so that node provisioning will work for these new nodes too. The Base View and `cmsh` front ends for creating new node objects and properties in CMDaemon are described.

Section 5.8 describes troubleshooting the node provisioning process.

5.1 Before The Kernel Loads

Immediately after powering up a node, and before it is able to load up the Linux kernel, a node starts its boot process in several possible ways:

5.1.1 PXE Booting

By default, nodes boot from the network when using BCM. This is called a *network boot*. On the `x86_64` (`amd64`) architecture it is known as a *PXE boot* (often pronounced as "pixie boot"). It is recommended as a BIOS setting for nodes. The head node runs a tftpd server that is managed by `systemd`. The tftpd server supplies the boot loader from within the default software image (section 2.1.2) offered to nodes.

The boot loader runs on the node and displays a menu (figure 5.1) based on loading a menu module within a configuration file. The default configuration files offered to nodes are located under `/tftpboot/pxelinux.cfg/` on the head node. To implement changes in the files, CMDaemon may need to be restarted, or the `updateprovisioners` command (page 250) can be run.

The default configuration files give instructions to the menu module of `PXElinux`. The instruction set used is documented at `http://www.syslinux.org/wiki/index.php/Comboot/menu.c32`, and includes the `TIMEOUT`, `LABEL`, `MENU LABEL`, `DEFAULT`, and `MENU DEFAULT` instructions.

The PXE `TIMEOUT` Instruction

During the display of the PXE boot menu, a selection can be made within a timeout period to boot the node in a several ways. Among the options are some of the install mode options (section 5.4.4). If no

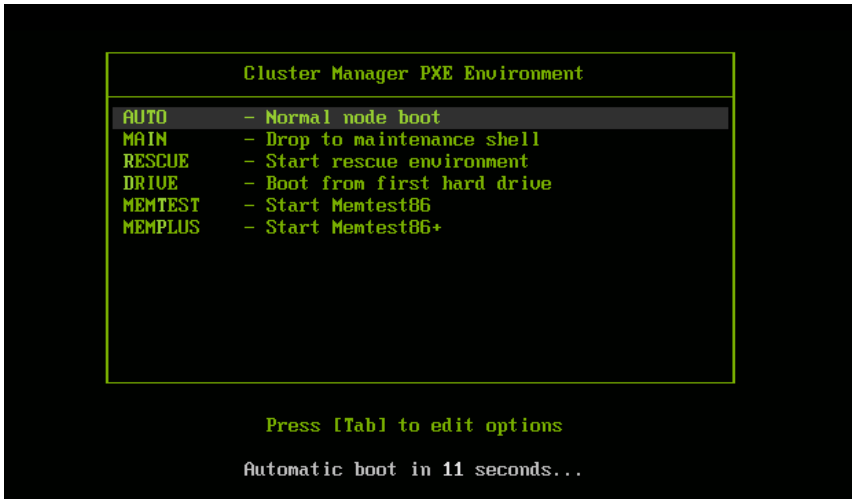


Figure 5.1: PXE boot menu options

selection is made by the user within the timeout period, then the AUTO install mode option is chosen by default.

In the PXE menu configuration files under pxelinux.cfg/, the default timeout of 5 seconds can be adjusted by changing the value of the "TIMEOUT 50" line. This value is specified in deciseconds.

Example

```
TIMEOUT 300   # changed timeout from 50 (=5 seconds)
```

The PXE LABEL And MENU LABEL Instructions

LABEL: The menu configuration files under pxelinux.cfg/ contain several multiline LABEL statements.

Each LABEL statement is associated with a kernel image that can be loaded from the PXE boot menu along with appropriate kernel options.

Each LABEL statement also has a text immediately following the LABEL tag. Typically the text is a description, such as linux, main, RESCUE, and so on. If the PXE menu module is not used, then tab completion prompting displays the list of possible text values at the PXE boot prompt so that the associated kernel image and options can be chosen by user intervention.

MENU LABEL: By default, the PXE menu module is used, and by default, each LABEL statement also contains a MENU LABEL instruction. Each MENU LABEL instruction also has a text immediately following the MENU LABEL tag. Typically the text is a description, such as AUTO, RESCUE and so on (figure 5.1). Using the PXE menu module means that the list of the MENU LABEL text values is displayed when the PXE boot menu is displayed, so that the associated kernel image and options can conveniently be selected by user intervention.

The PXE DEFAULT And MENU DEFAULT Instructions

DEFAULT: If the PXE menu module is not used and if no MENU instructions are used, and if there is no user intervention, then setting the same text that follows a LABEL tag immediately after the DEFAULT instruction, results in the associated kernel image and its options being run by default after the timeout.

By default, as already explained, the PXE menu module is used. In particular it uses the setting: DEFAULT menu.c32 to enable the menu.

MENU DEFAULT: If the PXE menu module is used and if MENU instructions are used, and if there is no user intervention, then setting a MENU DEFAULT tag as a line within the multiline LABEL statement results in the kernel image and options associated with that LABEL statement being loaded by default after the timeout.

The CMDaemon `PXE Label` Setting For Specific Nodes

The `MENU DEFAULT` value by default applies to every node using the software image that the PXE menu configuration file under `pxelinux.cfg/` is loaded from. To override its application on a per-node basis, the value of `PXE Label` can be set for each node.

- Some simple examples of overriding the default `MENU DEFAULT` value are as follows:
  - For example, using `cmsh`:

    Example

    ```
    [root@basecm11 ~]# cmsh
    [basecm11]% device use node001
    [basecm11->device[node001]]% set pxelabel MEMTEST ; commit
    ```

    Carrying it out for all nodes in the `default` category can be done, for example, with:

    Example

    ```
    [root@basecm11 ~]# cmsh
    [basecm11]% device
    [basecm11->device]% foreach -c default (set pxelabel MEMTEST)
    [basecm11->device*]% commit
    ```

    The value of `pxelabel` can be cleared with:

    Example

    ```
    [root@basecm11 ~]# cmsh -c "device; foreach -c default (clear pxelabel); commit"
    ```

  - In Base View, the PXE label can be set for a node `node001` using the navigation path `Devices > All Devices > Physical Node[node001]` > `Settings > Provisioning > PXE Label`
    which leads to a screen as in (figure 5.2):

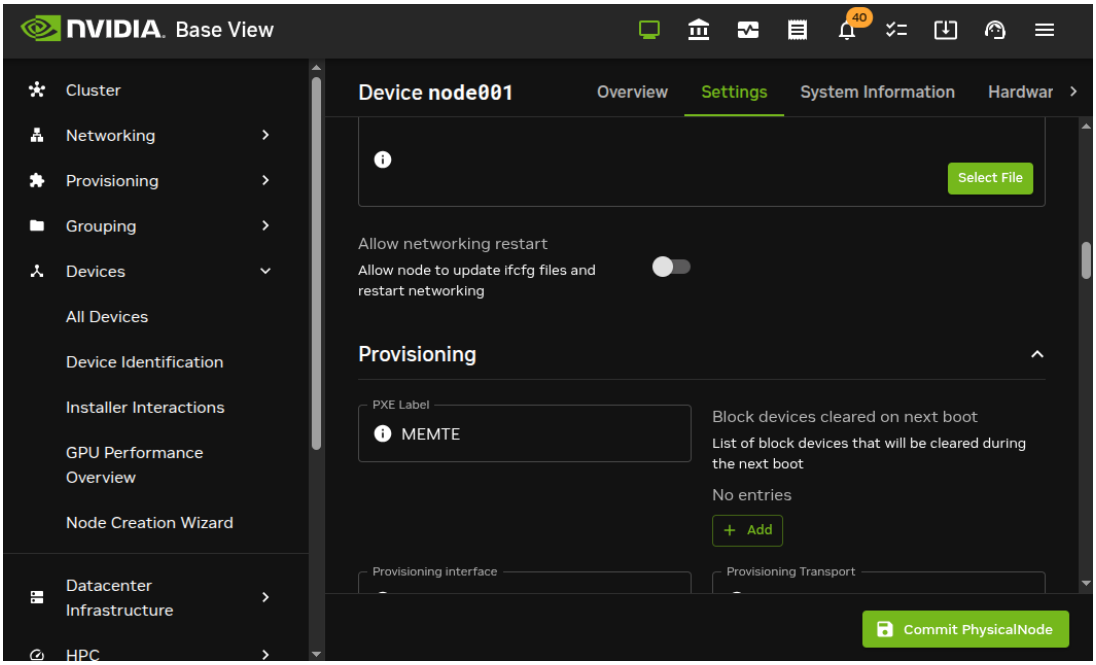


Figure 5.2: Base View `PXE Label` option

- A more complicated example of overriding the default `MENU DEFAULT` value now follows. Although it helps in understanding how PXE labels can be used, it can normally be skipped because the use case for it is unlikely, and the details are involved.

  In this example, `pxelabel` is set by the administrator using Base View or `cmsh` to `localdrive`. This will then set the node to boot from the first local drive and not the node-installer. This is a setting that is discouraged since it usually makes node management harder, but it can be used by administrators who do not wish to answer any prompt during node boot, and also want the node drives to have no risk of being overwritten by the actions of the node-installer, and also want the system to be up and running quite fully, even if not necessarily provisioned with the latest image from the head node.

  Here, the overwriting-avoidance method relies on the nodes being associated with a configuration file under `pxelinux.cfg` at the time that the `localdrive` setting is done. However, nodes that are unidentified, or are identified later on, will have their `MENU DEFAULT` value still set to a default `pxelabel` value set in the files under `/tftpboot/pxelinux.cfg/`, which is the value `linux` by default, and which is associated with a code block in that file with the label `LABEL linux`. To make such (as yet) unidentified nodes boot to a `localdrive` setting instead, requires modifying the files under `/tftpboot/pxelinux.cfg/`, so that the `MENU DEFAULT` line is associated with the code block of `LABEL localdrive` rather than the code block of `LABEL linux`.

  There are two methods other than using the preceding `pxelabel` method to deal with the risk of overwriting. Unlike the `pxelabel` method however, these methods can interrupt node booting, so that the node does not progress to being fully up until the administrator takes further action:

  1. If it is acceptable that the administrator manually enters a confirmation as part of the boot process when a possible overwrite risk is found, then the `datanode` method (section 5.4.4) can be used.
  2. If it is acceptable that the boot process halts on detecting a possible overwrite risk, then the XML assertions method (Appendix D.11) is recommended.

Changing The Install Mode Or Default Image Offered To Nodes

The selections offered by the PXE menu are pre-configured by default so that the `AUTO` menu option by default loads a kernel, runs the `AUTO` install mode, and eventually the `default-image` software image is provisioned.

Normally administrators should not be changing the install mode, kernel, or kernel options in the PXE menu configuration files under `pxelinux.cfg/`.

More on changing the install mode is given in section 5.4.4. More on changing software images, image package management, kernels, and kernel options, is to be found in Chapter 9.

5.1.2 iPXE Booting From A Disk Drive

Also by default, on disked nodes, iPXE software is placed on the drive during node installation. If the boot instructions from the BIOS for PXE booting fail, and if the BIOS instructions are that a boot attempt should then be made from the hard drive, it means that a PXE network boot attempt is done again, as instructed by the bootable hard drive. This can be a useful fallback option that works around certain BIOS features or problems.

5.1.3 iPXE Booting Using InfiniBand

On clusters that have InfiniBand hardware, it is normally used for data transfer as a service after the nodes have fully booted up (section 3.6). InfiniBand can also be used for PXE booting (described here) and used for node provisioning (section 5.3.3). However these uses are not necessary, even if InfiniBand is used for data transfer as a service later on, because booting and provisioning is available over Ethernet by default. This section (about boot over InfiniBand) may therefore safely be skipped when first configuring a cluster.

Booting over InfiniBand using PXE is enabled by carrying out these 3 steps:

1. Making BCM aware that nodes are to be booted over InfiniBand. Node booting (section 3.2.3, page 115) can be set from `cmsh` or Base View as follows:

   (a) From `cmsh`'s `network` mode: If the InfiniBand network name is `ibnet`, then a `cmsh` command that will set it is:
   ```
   cmsh -c "network; set ibnet nodebooting yes; commit"
   ```

   (b) From Base View: The `Settings` window for the InfiniBand network, for example `ibnet`, can be accessed from the `Networking` resource using the navigation path:

   `Networking > Networks > ibnet`  `> Settings`

   This is similar to figure 3.5, but for `ibnet`. The `Node booting` option for `ibnet` is then enabled and saved.

   If the InfiniBand network does not yet exist, then it must be created (section 3.2.2). The recommended default values used are described in section 3.6.4. The MAC address of the interface in CMDaemon defaults to using the GUID of the interface.

   The administrator should also be aware that the interface from which a node boots, (conveniently labeled `BOOTIF`), must not be an interface that is already configured for that node in CMDaemon. For example, if `BOOTIF` is the device `ib0`, then `ib0` must not already be configured in CMDaemon. Either `BOOTIF` or the `ib0` configuration should be changed so that node installation can succeed. It is recommended to set `BOOTIF` to `eth0` if the `ib0` device should exist.

2. Flashing iPXE onto the InfiniBand HCAs. (The ROM image is obtained from the HCA vendor).

3. Configuring the BIOS of the nodes to boot from the InfiniBand HCA.

All MAC addresses become invalid for identification purposes when changing from booting over Ethernet to booting over InfiniBand.

Administrators who enable iPXE booting almost always wish to provision over InfiniBand too. Configuring provisioning over InfiniBand is described in section 5.3.3.

5.1.4 Using PXE To Boot From The Drive

Besides PXE booting from only the network, a node can also be configured using PXE to step over to using its own drive to start booting and get to the stage of loading up its kernel entirely from its drive, just like a normal standalone machine. This can be done by setting `PXE LABEL` to `localdrive` (page 239).

5.1.5 Network Booting Without PXE On The ARMv8 Architecture

ARMv8 nodes use a network boot implementation that differs slightly from the x86 PXE boot implementation. The actual firmware that starts up on ARMv8 nodes depends on the environment the hardware is in. Networking is then started by the firmware and the network requests what to boot. The head node however then sends out a GRUB binary instead of an iPXE binary. The GRUB binary then runs on the regular node, and fetches the kernel and initrd using TFTP. The node-installer then runs on the nodes and follows the same steps as in the x86 process.

5.1.6 Network Booting Protocol

The protocol used by network booting is set with the parameter `bootloaderprotocol`. It is set to HTTP by default at category level:

Example

```
[basecm11->category[mydefaultimage]]% get bootloaderprotocol
HTTP
```

The protocol can be modified at category or node level, to one of the values HTTP, HTTPS, or TFTP:

Example

```
[basecm11->device[node001]]% get bootloaderprotocol
HTTP (mydefaultimage)
[basecm11->device[node001]]% set bootloaderprotocol <TAB><TAB>
http https tftp
[basecm11->device[node001]]% set bootloaderprotocol tftp
[basecm11->device*[node001*]]% commit
[basecm11->device[node001]]% get bootloaderprotocol
TFTP
```

If the protocol has the setting of HTTP, then initial PXE booting actually still uses the TFTP protocol on x86_64 hardware. However, a switchover to the HTTP protocol is done when loading up the kernel and RAM disk.

The HTTPS protocol for node booting should almost never be used, because it is rarely implemented in hardware.

5.1.7 The Boot Role

The action of providing a boot image to a node using DHCP and TFTP is known as providing *node booting*. Node provisioning (section 5.2), on the other hand, is about provisioning the node with the rest of the node image.

Roles in general are introduced in section 2.1.5. The *boot role* is one such role that can be assigned to a regular node. The boot role configures a regular node so that it can then provide node booting. The role cannot be assigned or removed from the head node—the head node always has a boot role.

The boot role is assigned by administrators to regular nodes if there is a need to cope with the scaling limitations of TFTP and DHCP. TFTP and DHCP services can be overwhelmed when there are large numbers of nodes making use of them during boot. An example of the scaling limitations may be observed, for example, when, during the powering up and network booting attempts of a large number of regular nodes from the head node, it turns out that random different regular nodes are unable to boot, typically due to network effects.

One implementation of boot role assignment might therefore be, for example, to have a several groups of racks, with each rack in a subnet, and with one regular node in each subnet that is assigned the boot role. The boot role regular nodes would thus take the DHCP and TFTP load off the head node and onto themselves for all the nodes in their associated subnet, so that all nodes of the cluster are then able to boot without networking issues.

5.2 Provisioning Nodes

The action of transferring the software image to the nodes is called *node provisioning*, and is done by special nodes called the *provisioning nodes*. More complex clusters can have several provisioning nodes configured by the administrator, thereby distributing network traffic loads when many nodes are booting.

Creating provisioning nodes is done by assigning a *provisioning role* to a node or category of nodes. Similar to how the head node always has a boot role (section 5.1.7), the head node also always has a provisioning role.

5.2.1 Provisioning Nodes: Configuration Settings

The provisioning role has several parameters that can be set:

Property	Description
allImages	The following values decide what images the provisioning node provides: • onlocaldisk (the default): all images on the local disk, regardless of any other parameters set • onlocaldiskexceptsharedimages: all images on the local disk, except for shared images • onsharedstorage: all images on the shared storage, regardless of any other parameters set • no: only images listed in the localimages or sharedimages parameters, described next
localimages	A list of software images on the local disk that the provisioning node accesses and provides. The list is used only if allImages is "no".
sharedimages	A list of software images on the shared storage that the provisioning node accesses and provides. The list is used only if allImages is "no"
Provisioning slots	The maximum number of nodes that can be provisioned in parallel by the provisioning node. The optimum number depends on the infrastructure. The default value is 10, which is safe for typical cluster setups. Setting it lower may sometimes be needed to prevent network and disk overload.
nodegroups	A list of node groups (section 2.1.4). If set, the provisioning node only provisions nodes in the listed groups. Conversely, nodes in one of these groups can only be provisioned by provisioning nodes that have that group set. Nodes without a group, or nodes in a group not listed in nodegroups, can only be provisioned by provisioning nodes that have no nodegroups values set. By default, the nodegroups list is unset in the provisioning nodes. The nodegroups setting is typically used to set up a convenient hierarchy of provisioning, for example based on grouping by rack and by groups of racks.
localprovisioning	If set to yes (the default), then it speeds up the initial provisioning of cloud directors and cloud provisioning nodes (section 3.2 of the *Cloudbursting Manual*) . Also if set to yes, if a software image is used as the rootfs of the provisioning node and is also to be used by that node to provision other cloud nodes, then during the initial FULL install (page 276) that image is transferred only once to the provisioning node, instead of twice.

A provisioning node keeps a copy of all the images it provisions on its local drive, in the same directory as where the head node keeps such images. The local drive of a provisioning node must therefore have enough space available for these images, which may require changes in its disk layout.

5.2.2 Provisioning Nodes: Role Setup With cmsh

In the following cmsh example the administrator creates a new category called misc. The default category default already exists in a newly installed cluster.

The administrator then assigns the role called provisioning, from the list of available assignable roles, to nodes in the misc category. After the assign command has been typed in, but before entering

the command, tab-completion prompting can be used to list all the possible roles. Assignment creates an association between the role and the category. When the `assign` command runs, the shell drops into the level representing the `provisioning` role.

If the role called `provisioning` were already assigned, then the `use provisioning` command would drop the shell into the `provisioning` role, without creating the association between the role and the category.

As an aside from the topic of provisioning, from an organizational perspective, other assignable roles include `monitoring`, `storage`, and `failover`.

Once the shell is within the role level, the role properties can be edited conveniently.

For example, the nodes in the `misc` category assigned the `provisioning` role can have `default-image` set as the image that they provision to other nodes, and have 20 set as the maximum number of other nodes to be provisioned simultaneously (some text is elided in the following example):

Example

```
[basecm11]% category add misc
[basecm11->category*[misc*]]% roles
[basecm11->category*[misc*]->roles]% assign provisioning
[basecm11...*]->roles*[provisioning*]]% set allimages no
[basecm11...*]->roles*[provisioning*]]% set localimages default-image
[basecm11...*]->roles*[provisioning*]]% set provisioningslots 20
[basecm11...*]->roles*[provisioning*]]% show
Parameter                        Value
-------------------------------- --------------------------------
Name                             provisioning
Revision
Type                             ProvisioningRole
Add services                     yes
Provisioning slots               20
All images                       no
Include revisions of local images yes
Local provisioning               yes
Local images                     default-image
Shared images
Nodegroups
Categories
Racks
Load weight                      0.000000
[basecm11->category*[misc*]->roles*[provisioning*]]% commit
[basecm11->category[misc]->roles[provisioning]]%
```

Assigning a `provisioning` role can also be done for an individual node instead, if using a category is deemed overkill:

Example

```
[basecm11]% device use node001
[basecm11->device[node001]]% roles
[basecm11->device[node001]->roles]% assign provisioning
[basecm11->device*[node001*]->roles*[provisioning*]]%
...
```

A role change configures a provisioning node, but does not directly update the provisioning node with images. After carrying out a role change, BCM runs the `updateprovisioners` command described in section 5.2.4 automatically, so that regular images are propagated to the provisioners. The propagation can be done by provisioners themselves if they have up-to-date images. CMDaemon tracks the

provisioning nodes role changes, as well as which provisioning nodes have up-to-date images available, so that provisioning node configurations and regular node images propagate efficiently. Thus, for example, image update requests by provisioning nodes take priority over provisioning update requests from regular nodes.

5.2.3 Provisioning Nodes: Role Setup With Base View

The provisioning configuration outlined in `cmsh` mode in section 5.2.2 can be done using Base View too, as follows:

A `misc` category can be added using the navigation path

`Grouping > Categories > + Add > Settings` <*name*>

That is, within the `Settings` tab, the node category should be given a name `misc` (figure 5.3), and saved:

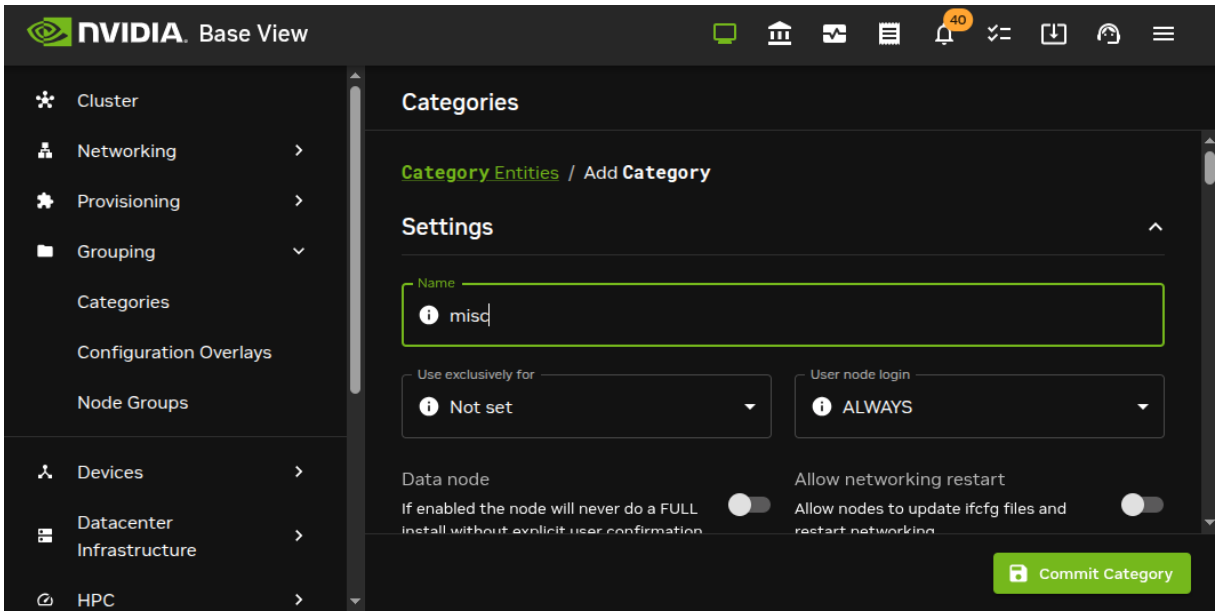


Figure 5.3: Base View: Adding A `misc` Category

The `Roles` window can then be opened by using the relative navigation path from the `misc` category pane:

`...  > Roles`  `> + Add`

A scrollable list of available roles is then displayed, (figure 5.4):

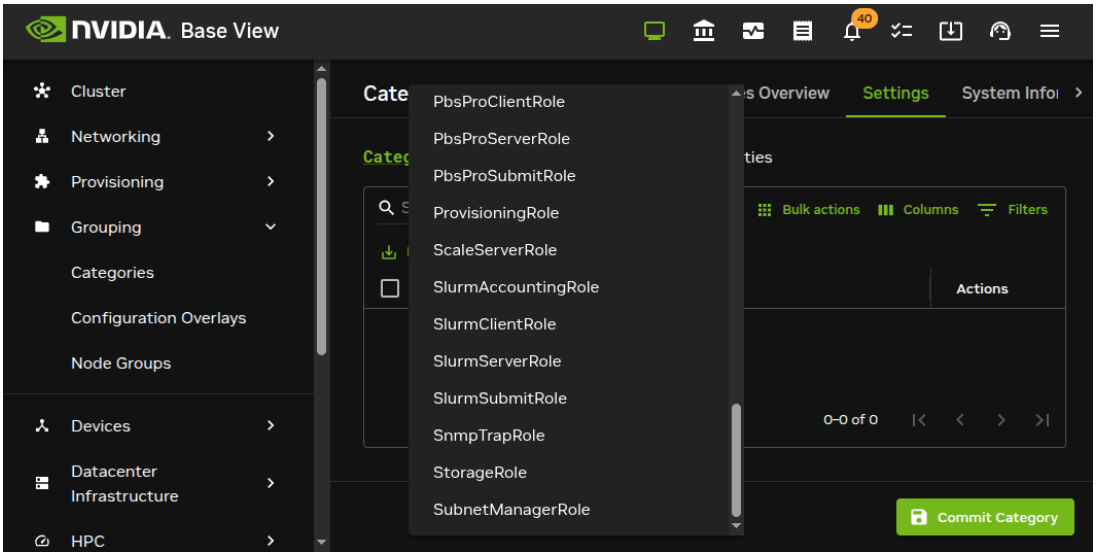


Figure 5.4: Base View: Setting A `provisioning` Role

If the category `misc` already exists, then the equivalent navigation path would be

`Grouping > Categories > misc` > `Settings Roles` > `+ Add`

to display the same scrollable list of available roles (figure 5.4).

After selecting a role, the role can be saved using the `Commit Category` button there. A warning is displayed if the configuration properties have any issues. The warning can be examined by clicking through (figure 5.5):

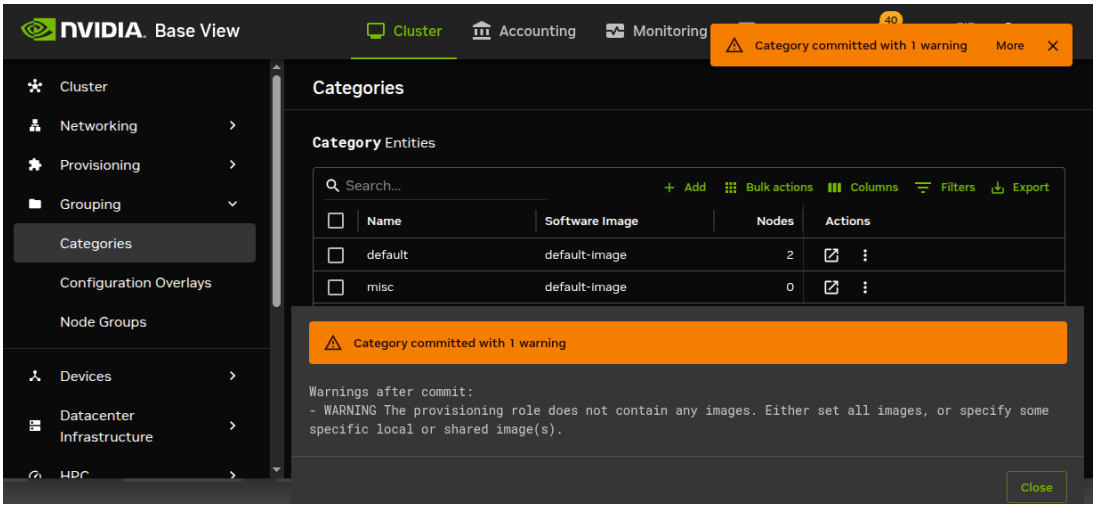


Figure 5.5: Base View: Warning When Setting A `provisioning` Role

The configuration properties of the role can be edited. To access the `provisioning` role properties in the `misc` category, the full navigation path would be:

`Grouping > Categories > misc` > `Settings Roles` > `provisioning`

This brings up a pane where values can be modified (figure 5.6):

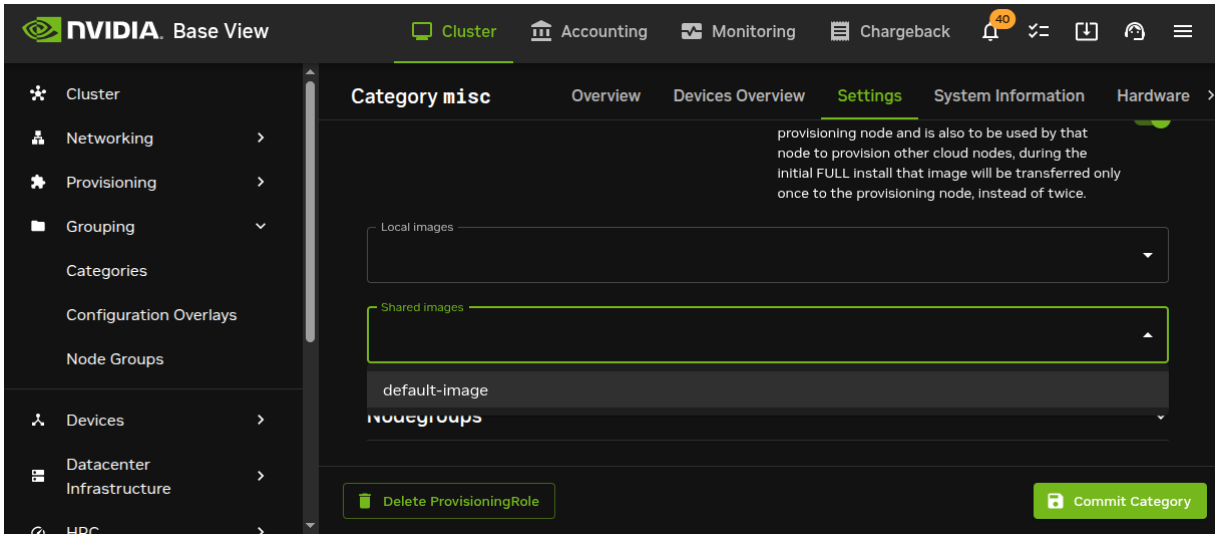


Figure 5.6: Base View: Configuring A `provisioning` Role

For example:

- the `Provisioning slots` setting decides how many images can be supplied simultaneously from the provisioning node
- the `All images` setting decides if the role provides all images
- the `Local images` setting decides what images the provisioning node supplies from local storage
- the `Shared images` setting decides what images the provisioning node supplies shared storage.

The settings can be saved with the `Commit Category` button of figure 5.6. The images offered by the provisioning role in the navigation path:

`Grouping > Categories > misc` > `Settings Roles` > `provisioning` > *local or shared software image for provisioning*

should not be confused with the software image setting of the `misc` category itself, which is the image the provisioning node requests for itself from the category, as seen in the navigation path:

`Grouping > Categories > misc` > `Settings > Software image` [*software image for category*]

5.2.4 Provisioning Nodes: Housekeeping

The head node does housekeeping tasks for the entire provisioning system. Provisioning is done on request for all non-head nodes on a first-come, first-serve basis. Since provisioning nodes themselves, too, need to be provisioned, it means that to cold boot an entire cluster up quickest, the head node should be booted and be up first, followed by provisioning nodes, and finally by all other non-head nodes. Following this start-up sequence ensures that all provisioning services are available when the other non-head nodes are started up.

Some aspects of provisioning housekeeping are discussed next:

Provisioning Node Selection

When a node requests provisioning, the head node allocates the task to a provisioning node. If there are several provisioning nodes that can provide the image required, then the task is allocated to the provisioning node with the lowest number of already-started provisioning tasks.

Limiting Provisioning Tasks With MaxNumberOfProvisioningThreads

Besides limiting how much simultaneous provisioning per provisioning node is allowed with Provisioning Slots (section 5.2.1), the head node also limits how many simultaneous provisioning tasks are allowed to run on the entire cluster. This is set using the MaxNumberOfProvisioningThreads directive in the head node's CMDaemon configuration file, /etc/cmd.conf, as described in Appendix C.

Provisioning Tasks Deferral and Failure

A provisioning request is *deferred* if the head node is not able to immediately allocate a provisioning node for the task. Whenever an ongoing provisioning task has finished, the head node tries to re-allocate deferred requests.

A provisioning request *fails* if an image is not transferred. 5 retry attempts at provisioning the image are made in case a provisioning request fails.

A provisioning node that is carrying out requests, and which loses connectivity, has its provisioning requests remain allocated to it for 180 seconds from the time that connectivity was lost. After this time the provisioning requests fail.

Provisioning Role Change Notification With updateprovisioners

The updateprovisioners command is used to update the image on the provisioning nodes.

It can be accessed from the softwareimage mode in cmsh.

It can also be accessed from Base View, using the navigation path:

Software Images > ⋮ > Provisioning > Update provisioner

In the examples in section 5.2.2, changes were made to provisioning role attributes for an individual node as well as for a category of nodes. This automatically ran the updateprovisioners command.

The updateprovisioners command runs automatically if CMDaemon is involved during software image changes or during a provisioning request. If on the other hand, the software image is changed outside of the CMDaemon front ends (Base View and cmsh), for example by an administrator adding a file by copying it into place from the bash prompt, then updateprovisioners should be run manually to update the provisioners.

In any case, if it is not run manually, then by default it runs every midnight (UTC). The scheduling period can be adjusted with the autoupdateperiod setting:

Example

```
[root@basecm11 ]# cmsh
[basecm11]% partition use base
[basecm11->partition[base]]% provisioningsettings
[basecm11->partition[base]->provisioningsettings]% get autoupdateperiod
[basecm11->partition[base]->provisioningsettings]% 86400
[basecm11->partition[base]->provisioningsettings]% # is UTC epoch start time modulo 24 hours
[basecm11->partition[base]->provisioningsettings]% # set UTC epoch start time modulo 18 hours:
[basecm11->partition[base]->provisioningsettings]% set autoupdateperiod 64800
[basecm11->partition*[base*]->provisioningsettings*]% commit
```

When the default updateprovisioners is invoked manually, the provisioning system waits for all running provisioning tasks to end, and then updates all images located on any provisioning nodes by using the images on the head node. It also re-initializes its internal state with the updated provisioning role properties, i.e. keeps track of what nodes are provisioning nodes.

The default updateprovisioners command, run with no options, updates all images. If run from cmsh with a specified image as an option, then the command only does the updates for that particular image. A provisioning node undergoing an image update does not provision other nodes until the update is completed.

Example

```
[basecm11]% softwareimage updateprovisioners
Provisioning nodes will be updated in the background.

Sun Dec 12 13:45:09 2010 basecm11: Starting update of software image(s)\
 provisioning node(s). (user initiated).
[basecm11]% softwareimage updateprovisioners [basecm11]%
Sun Dec 12 13:45:41 2010 basecm11: Updating image default-image on prov\
isioning node node001.
[basecm11]%
Sun Dec 12 13:46:00 2010 basecm11: Updating image default-image on prov\
isioning node node001 completed.
Sun Dec 12 13:46:00 2010 basecm11: Provisioning node node001 was updated
Sun Dec 12 13:46:00 2010 basecm11: Finished updating software image(s) \
on provisioning node(s).
```

Provisioning Role Draining And Undraining Nodes With `drain`, `undrain`

The `drain` and `undrain` commands to control provisioning nodes are accessible from within the `softwareimage` mode of `cmsh`.

If a node is put into a `drain` state, then all currently active provisioning requests continue until they are completed. However, the node is not assigned any further pending requests, until the node is put back into an `undrain` state.

Example

```
[basecm11->softwareimage]% drain -n master
Nodes drained
[basecm11->softwareimage]% provisioningstatus
Provisioning subsystem status
Pending request:            node001, node002
Provisioning node status:
+ basecm11
  Slots:                    1 / 10
  State:                    draining
  Active nodes:             node003
  Up to date images:        default-image
[basecm11->softwareimage]% provisioningstatus
Provisioning subsystem status
Pending request:            node001, node002
Provisioning node status:
+ basecm11
  Slots:                    0 / 10
  State:                    drained
  Active nodes:             none
  Up to date images:        default-image
```

To drain all nodes at once, the `--role` option can be used, with `provisioning` role as its value. All pending requests then remain in the queue, until the nodes are undrained again.

Example

```
[basecm11->softwareimage]% drain --role provisioning
...Time passes. Pending
   requests stay in the queue. Then
   admin undrains it...
[basecm11->softwareimage]% undrain --role provisioning
```

Provisioning Node Update Safeguards And `dirtyautoupdatetimeout`

The `updateprovisioners` command is subject to safeguards that prevent it running too frequently. The minimum period between provisioning updates can be adjusted with a timeout parameter `dirtyautoupdatetimeout`, which has a default value of 300s.

Exceeding the timeout does not by itself trigger an update to the provisioning node.

When the head node receives a provisioning request, it checks if the last update of the provisioning nodes is more than the timeout period. If true, then an update is triggered to the provisioning node. The update is disabled if the `dirtyautoupdatetimeout` is set to zero (`false`).

The parameter can be accessed and set within `cmsh` from `partition` mode:

Example

```
[root@basecm11 ]# cmsh
[basecm11]% partition use base
[basecm11->partition[base]]% provisioningsettings
[basecm11->partition[base]->provisioningsettings]% get dirtyautoupdatetimeout
[basecm11->partition[base]->provisioningsettings]% 300
[basecm11->partition[base]->provisioningsettings]% set dirtyautoupdatetimeout 0
[basecm11->partition*[base*]->provisioningsettings*]% commit
```

Within Base View the parameter is accessible using the navigation path:

`Cluster > Settings > BMC Settings > Provisioning Settings >`  `> Dirty auto update timeout`

To prevent provisioning an image to the nodes, it can be locked (section 5.4.7). The provisioning request is then deferred until the image is once more unlocked.

Synchronization Of Fspart Subdirectories To Provisioning Nodes

In BCM, an *fspart* is a subdirectory, and it is a filesystem part that can be synced during provisioning.

The fsparts can be listed with:

Example

```
[root@basecm11 ]# cmsh
[basecm11]% fspart
[basecm11->fspart]% list
Path (key)                    Type            Image
----------------------------- --------------- ------------------------
/cm/images/default-image      image           default-image
/cm/images/default-image/boot boot            default-image:boot
/cm/node-installer            node-installer
/cm/shared                    cm-shared
/tftpboot                     tftpboot
/var/spool/cmd/monitoring     monitoring
```

The `updateprovisioners` command (page 250) is used to update image fsparts to all nodes with a provisioning role.

The `trigger` command: is used to update non-image fsparts to off-premises nodes, such as cloud directors and edge directors. The directors have a provisioning role for the nodes that they direct.

All of the non-image types can be updated with the `--all` option:

Example

```
[basecm11->fspart]% trigger --all
```

The command `help trigger` in `fspart` mode gives further details.

The `info` command: shows the architecture, OS, and the number of inotify watchers that track rsyncs in the fspart subdirectory.

```
[basecm11->fspart]% info
Path                          Architecture     OS               Inotify watchers
----------------------------- ---------------- ---------------- ----------------
/cm/images/default-image      x86_64           rhel9            0
/cm/images/default-image/boot -                -                0
/cm/node-installer            x86_64           rhel9            0
/cm/shared                    x86_64           rhel9            0
/tftpboot                     -                -                0
/var/spool/cmd/monitoring     -                -                0

[basecm11->fspart]% info -s                        (!#with size, takes longer)
Path                          Architecture     OS               Inotify watchers Size
----------------------------- ---------------- ---------------- ---------------- ----------------
/cm/images/default-image      x86_64           rhel9            0                4.8 GiB
/cm/images/default-image/boot -                -                0                313 MiB
/cm/node-installer            x86_64           rhel9            0                2.84 GiB
/cm/shared                    x86_64           rhel9            0                1.16 GiB
/tftpboot                     -                -                0                3.5 MiB
/var/spool/cmd/monitoring     -                -                0                1.02 GiB
```

The `locked`, `lock`, and `unlock` commands:

- The `locked` command lists fsparts that are prevented from syncing.

  Example

  ```
  [basecm11->fspart]% locked
  No locked fsparts
  ```

- The `lock` command prevents a specific fspart from syncing.

  Example

  ```
  [basecm11->fspart]% lock /var/spool/cmd/monitoring
  [basecm11->fspart]% locked
  /var/spool/cmd/monitoring
  ```

- The `unlock` command unlocks a specific locked fspart again.

  Example

  ```
  [basecm11->fspart]% unlock /var/spool/cmd/monitoring
  [basecm11->fspart]% locked
  No locked fsparts
  ```

Access to `excludelistsnippets`: The properties of `excludelistsnippets` for a specific fspart can be accessed from the `excludelistsnippets` submode:

Example

```
[basecm11->fspart]% excludelistsnippets /tftpboot

[basecm11->fspart[/tftpboot]->excludelistsnippets]% list
Name (key)   Lines   Disabled     Mode sync   Mode full   Mode update    Mode grab   Mode grab new
------------ ------- ------------ ----------- ----------- -------------- ----------- --------------
Default      2       no           yes         yes         yes            no          no

[basecm11->fspart[/tftpboot]->excludelistsnippets]% show default
Parameter                      Value
------------------------------ ------------------------------------------------------------------
Lines                          2
Name                           Default
Revision
Exclude list                   # no need for rescue on nodes with a boot role,/rescue,/rescue/*
Disabled                       no
No new files                   no
Mode sync                      yes
Mode full                      yes
Mode update                    yes
Mode grab                      no
Mode grab new                  no

[basecm11->fspart[/tftpboot]->excludelistsnippets]% get default excludelist
# no need for rescue on nodes with a boot role
/rescue
/rescue/*
```

Last Provisioning Node: `lastprovisioningnode`

The `lastprovisioningnode` command is run from within `device` mode. It displays a list of nodes, along with the provisioning server that last provisioned each node. The `lastprovisioningnode` command can be useful for tracking which provisioning nodes are handling the provisioning workload, and for troubleshooting provisioning issues.

Example

```
[basecm11->device]% lastprovisioningnode --inverse
Node                     Server
------------------------ ---------------
a06-u02-bcm-01
a06-u04-bcm-02
a06-u06-slurm-01         a06-u02-bcm-01
a06-u11-k8s-admin-01     a06-u02-bcm-01
a06-u13-k8s-admin-02     a06-u02-bcm-01
a06-u15-k8s-admin-03     a06-u02-bcm-01
a06-u17-k8s-run-01       a06-u02-bcm-01
a06-u19-k8s-run-02       a06-u02-bcm-01
a06-u21-k8s-run-03       a06-u02-bcm-01
a07-p01-dgx-02-c01       a06-u02-bcm-01
a07-p01-dgx-02-c02       a06-u02-bcm-01
a07-p01-dgx-02-c03       a06-u04-bcm-02
a07-p01-dgx-02-c04       a06-u02-bcm-01
...
```

Without the `--inverse` option, the output is grouped by provisioning server instead of by node (output truncated):

Example

```
[basecm11->device]% lastprovisioningnode
Server          Node
--------------- ------------------------------------------------------------
                a06-u02-bcm-01,a06-u04-bcm-02
a06-u02-bcm-01  a06-u06-slurm-01,a06-u17-k8s-run-01,a07-p01-dgx-02-c01,...
a06-u04-bcm-02  a07-p01-dgx-02-c03,a07-p01-dgx-02-c05,a07-p01-dgx-02-c07,...
```

The `lastprovisioningnode` command has options to filter by node, category, rack, and so on. Its help text (`help lastprovisioningnode`) has further details on its use.

Filesystem Partition Associations: `fspartassociation`

The `fspartassociation` command is run within `device` mode. It displays which filesystem partition paths are associated with each node, whether the node is a provider, uses shared storage, and any backup directory configurations. This is useful for understanding provisioning relationships and storage configurations:

Example

```
[basecm11->device]% fspartassociation
Hostname          Path                       Provider  Root  Shared storage  Backup directory
----------------- -------------------------- --------  ----  --------------  ----------------------
a06-u02-bcm-01    /cm/conf/all               yes       no    no
a06-u02-bcm-01    /cm/images/default-image   yes       no    no
a06-u02-bcm-01    /cm/images/dgx-image       yes       no    no
a06-u02-bcm-01    /tftpboot                  yes       no    no
a06-u02-bcm-01    /var/spool/cmd/monitoring  yes       no    no              /var/spool/cmd/backup
a06-u04-bcm-02    /cm/conf/all               yes       no    no
a06-u04-bcm-02    /cm/images/default-image   yes       no    no
...
a07-p01-dgx-02-c01  /cm/conf/all             no        no    no
a07-p01-dgx-02-c01  /cm/images/dgx-image     no        yes   no
...
```

The `fspartassociation` command has options to filter by node, category, rack, and so on. Its help text (`help fspartassociation`) gives further details on its use.

Aborting Provisioning With `cancelprovisioningrequest`

The `cancelprovisioningrequest` command is run from `softwareimage` mode, and cancels provisioning.

Its usage is:

```
cancelprovisioningrequest [OPTIONS] [<requestid> ...]
```

To cancel all provisioning requests, it can be run as:

```
basecm11:~ # cmsh -c "softwareimage cancelprovisioningrequest -a"
```

The `provisioningstatus` command of `cmsh`, can be used to find request IDs. Individual request IDs, for example `10` and `13`, can then be specified in the `cancelprovisioningrequest` command, as:

```
basecm11:~ # cmsh -c "softwareimage cancelprovisioningrequest 10 13"
```

The help page for `cancelprovisioningrequest` shows how to run the command on node ranges, groups, categories, racks, chassis, and so on.

5.3 The Kernel Image, Ramdisk And Kernel Modules

A *software image* is a complete Linux filesystem that is to be installed on a non-head node. Chapter 9 describes images and their management in detail.

The head node holds the head copy of the software images. Whenever files in the head copy are changed using CMDaemon, the changes automatically propagate to all provisioning nodes using the `updateprovisioners` command (section 5.2.4).

5.3.1 Booting To A "Good State" Software Image

When nodes boot from the network in simple clusters, the head node supplies them with a *known good state* during node start up. The known good state is maintained by the administrator and is defined using a software image that is kept in a directory of the filesystem on the head node. Supplementary filesystems such as `/home` are served using NFS from the head node by default.

For a diskless node the known good state is copied over from the head node, after which the node becomes available to cluster users.

For a disked node, by default, the hard disk contents on specified local directories of the node are checked against the known good state on the head node. Content that differs on the node is changed to that of the known good state. After the changes are done, the node becomes available to cluster users.

Each software image contains a Linux kernel and a ramdisk. These are the first parts of the image that are loaded onto a node during early boot. The kernel is loaded first. The ramdisk is loaded next, and contains driver modules for the node's network card and local storage. The rest of the image is loaded after that, during the node-installer stage (section 5.4).

5.3.2 Selecting Kernel Driver Modules To Load Onto Nodes

Kernel modules can be managed in `softwareimage` mode (using an image), in `category` mode (using a category), or in `device` mode (using a node), as indicated by the following `cmsh` tree view of a newly-installed cluster with default values:

```
cmsh
|-- category[default]
|   |-- kernelmodules
...
|-- device][node001]
|   |-- kernelmodules
...
|-- softwareimage[default-image]
|   |-- kernelmodules
...
```

As is usual in BCM, if there are values specified at the lower levels in the hierarchy, then their values override the values set higher up in the hierarchy. For example, modules specified at node level override modules specified at category or software image level. Similarly modules specified at the category level override whatever is specified at the software image level. The cluster administrators should be aware that "override" for kernel modules appended to a kernel image means that any kernel modules defined at a higher level are totally ignored—the modules from a lower level exclude the modules at the higher level. A misconfiguration of kernel modules in the lower levels can thus prevent the node from starting up.

Modules are normally just set at `softwareimage` level, using `cmsh` or Base View.

Kernel Driver Modules With `cmsh`

In `cmsh`, the modules that are to go on the ramdisk can be placed using the `kernelmodules` submode of the `softwareimage` mode. The order in which they are listed is the attempted load order.

Within the `kernelmodules` submode, the `import` command can be used to import the kernel modules list from:

- a software image (`-i|--image`)
- a node (`-n|--node`)
- a category (`-c|--category`)

The import replaces the original kernel modules list.

Example

```
[basecm11]% softwareimage use target-image
[basecm11->softwareimage[target-image]]% kernelmodules
[basecm11->softwareimage[target-image]->kernelmodules]% import -i default-image
54 module(s) successfully imported.
[basecm11->softwareimage[target-image]->kernelmodules*]% commit
```

Whenever a change is made using the `kernelmodules` submode to the kernel module selection of a software image, CMDaemon automatically runs the `createramdisk` command. The `createramdisk` command regenerates the ramdisk inside the initrd image and sends the updated image to all provisioning nodes, to the image directory, set by default to `/cm/images/default-image/boot/`. The original initrd image is saved as a file with suffix "`.orig`" in that directory. An attempt is made to generate the image for all software images that CMDaemon is aware of, regardless of category assignment, unless the image is protected from modification by CMDaemon with a `FrozenFile` directive (Appendix C).

The `createramdisk` command can also be run manually from within the `softwareimage` mode.

Kernel Driver Modules With Base View

In Base View the kernel modules for a particular image are managed through the `Software images` resource, and then choosing the `Kernel modules` menu option of that image. For example, for the image `default-image`, the navigation path that can be followed is:

`Provisioning > Software images[default-image] > Settings > Kernel modules`

which opens up the `KernelModule Entities` screen. Kernel modules can be removed or added in this screen (figure 5.7):

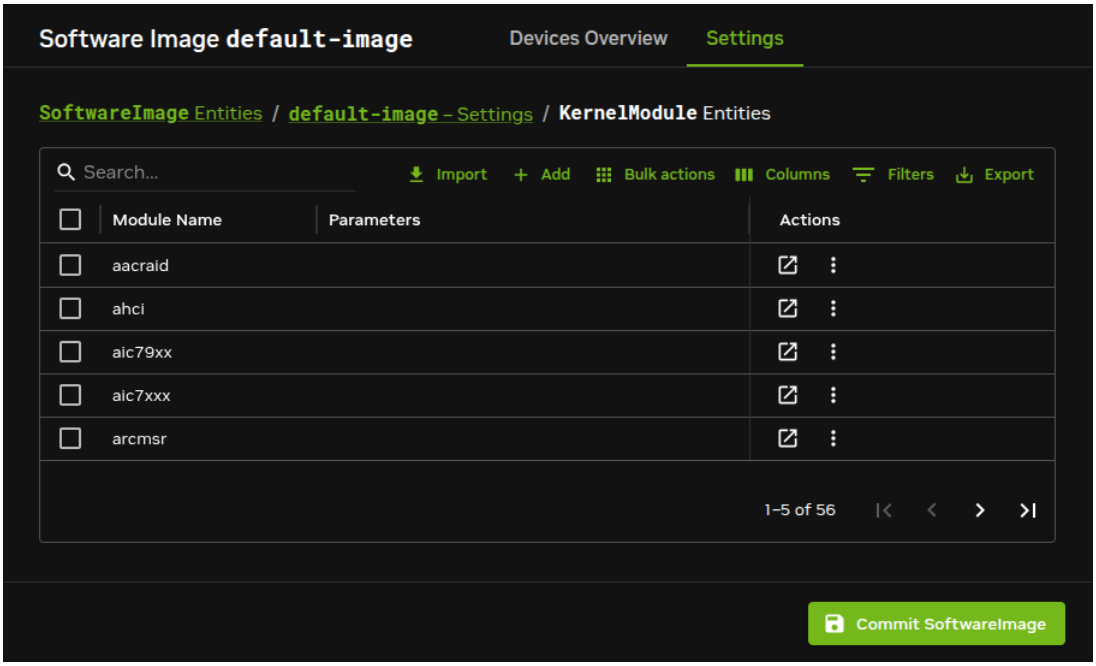


Figure 5.7: Base View: Selecting Kernel Modules For Software Images

The screen allows

- the + Add option to be used to add a new kernel module
- the ⋮ > Delete option to be used to remove a kernel module
- the ⧉ option to be used to set kernel module parameters

Manually Regenerating A Ramdisk

Regenerating a ramdisk manually using `cmsh` or Base View is useful if the kernel or modules have changed without using CMDaemon. For example, after running a YUM update which has modified the kernel or modules of the nodes (section 9.3). In such a case, the distribution would normally update the ramdisk on the machine, but this is not done for the extended ramdisk for nodes in BCM. Not regenerating the BCM ramdisk for nodes after such an update means the nodes may fail on rebooting during the loading of the ramdisk (section 5.8.4).

An example of regenerating the ramdisk is seen in section 5.8.6.

Implementation Of Kernel Driver Via Ramdisk Or Kernel Parameter

Sometimes, testing or setting a kernel driver as a kernel parameter may be more convenient. How to do that is covered in section 9.3.4.

5.3.3 InfiniBand Provisioning

On clusters that have InfiniBand hardware, it is normally used for data transfer as a service after the nodes have fully booted up (section 3.6). It can also be used for PXE booting (section 5.1.3) and for node provisioning (described here), but these are not normally a requirement. This section (about InfiniBand node provisioning) may therefore safely be skipped in almost all cases when first configuring a cluster.

During node start-up on a setup for which InfiniBand networking has been enabled, the `init` process runs the `rdma` script. For SLES the `openib` script is used instead of the `rdma` script. The script loads up InfiniBand modules into the kernel. When the cluster is finally fully up and running, the use of InfiniBand is thus available for all processes that request it.

Provisioning nodes over InfiniBand is not implemented by default, because the `init` process, which handles initialization scripts and daemons, takes place only after the node-provisioning stage launches. InfiniBand modules are therefore not available for use during provisioning, which is why, for default kernels, provisioning in BCM is done using Ethernet.

Provisioning at the faster InfiniBand speeds rather than Ethernet speeds is however a requirement for some clusters. To get the cluster to provision using InfiniBand requires both of the following two configuration changes to be carried out:

1. configuring InfiniBand drivers for the ramdisk image that the nodes first boot into, so that provisioning using InfiniBand is possible during this pre-`init` stage
2. defining the provisioning interface of nodes that are to be provisioned with InfiniBand. It is assumed that InfiniBand networking is already configured, as described in section 3.6.

   The administrator should be aware that the interface from which a node boots, (conveniently labeled `BOOTIF`), must not be an interface that is already configured for that node in CMDaemon. For example, if `BOOTIF` is the device `ib0`, then `ib0` must not already be configured in CMDaemon. Either `BOOTIF` or the `ib0` configuration should be changed so that node installation can succeed.

How these two changes are carried out is described next:

InfiniBand Provisioning: Ramdisk Image Configuration

An easy way to see what modules must be added to the ramdisk for a particular HCA can be found by running `rdma` (or `openibd`), and seeing what modules do load up on a fully booted regular node.

One way to do this is to run the following lines as root:

```
[root@basecm11 ~]# { service rdma stop; lsmod | cut -f1 -d" "; }>/tmp/a
[root@basecm11 ~]# { service rdma start; lsmod | cut -f1 -d" "; }>/tmp/b
```

The rdma service in the two lines should be replaced by openibd service instead when using SLES, or distributions based on versions of Red Hat prior to version 6.

The first line stops the InfiniBand service, just in case it is running, in order to unload its modules, and then lists the modules on the node.

The second line starts the service, so that the appropriate modules are loaded, and then lists the modules on the node again. The output of the first step is stored in a file a, and the output from the second step is stored in a file b.

Running diff on the output of these two steps then reveals the modules that get loaded. For rdma, the output may display something like:

Example

```
[root@basecm11 ~]# diff /tmp/a /tmp/b
1,3c1
< Unloading OpenIB kernel modules:
< Failed to unload ib_core
<                                                          [FAILED]
---
> Loading OpenIB kernel modules:                           [  OK  ]
4a3,14
> ib_ipoib
> rdma_ucm
> ib_ucm
> ib_uverbs
> ib_umad
> rdma_cm
> ib_cm
> iw_cm
> ib_addr
> ib_sa
> ib_mad
> ib_core
```

As suggested by the output, the modules ib_ipoib, rdma_ucm and so on are the modules loaded when rdma starts, and are therefore the modules that are needed for this particular HCA. Other HCAs may cause different modules to be loaded.

For a default Red Hat from version 7 onward, the rdma service can only be started; it cannot be stopped. Finding the modules that load can therefore only be done once for the default configuration, until the next reboot.

The preceding lsmod lines in that case can be generated with:

Example

```
[root@basecm11 ~]# { lsmod | cut -f1 -d" "; }>/tmp/a
[root@basecm11 ~]# { systemctl start rdma-load-modules@rdma; lsmod | cut -f1 -d" "; }>/tmp/b
```

Here, systemctl is used instead of the older service command just because it is the modern way to run such commands.

The InfiniBand modules that load are the ones that the initrd image needs, so that InfiniBand can be used during the node provisioning stage. The administrator can therefore now create an initrd image with the required InfiniBand modules.

Loading kernel modules into a ramdisk is covered in general in section 5.3.2. A typical Mellanox HCA may have an initrd image created as follows (some text ellipsized in the following example):

Example

```
[root@basecm11 ~]# cmsh
[basecm11]% softwareimage use default-image
[basecm11->softwareimage[default-image]]% kernelmodules
[basecm11...age[default-image]->kernelmodules]% add mlx4_ib
[basecm11...age*[default-image*]->kernelmodules*[mlx4_ib*]]% add ib_ipoib
[basecm11...age*[default-image*]->kernelmodules*[ib_ipoib*]]% add ib_umad
[basecm11...age*[default-image*]->kernelmodules*[ib_umad*]]% commit
[basecm11->softwareimage[default-image]->kernelmodules[ib_umad]]%
Tue May 24 03:45:35 2011 basecm11: Initial ramdisk for image default-image was regenerated successfully.
```

If the modules are put in another image instead of `default-image`, then the default image that nodes boot from should be set to the new image (section 3.20.2).

InfiniBand Provisioning: Network Configuration

It is assumed that the networking configuration for the final system for InfiniBand is configured following the general guidelines of section 3.6. If it is not, that should be checked first to see if all is well with the InfiniBand network.

The provisioning aspect is set by defining the provisioning interface. An example of how it may be set up for 150 nodes with a working InfiniBand interface `ib0` in `cmsh` is:

Example

```
[root@basecm11~]# cmsh
[basecm11]% device
[basecm11->device]% foreach -n node001..node150 (set provisioninginterface ib0)
[basecm11->device*]% commit
```

5.3.4 VLAN Provisioning

Nodes can be configured for provisioning over a VLAN interface, starting in NVIDIA Base Command Manager version 8.2.

This requires:

- A VLAN network and node interface. The VLAN network is typically specified by the network switch. The interface that connects the node to the switch can be configured as a VLAN interface as outlined in section 3.4.
- The `8021q` (`rtnl-link-vlan`) driver to be available in the software image that is provisioned. In recent distributions this driver is not part of the base kernel, and is instead available as a module. The module should be loaded into the software image that is to be provisioned. For example, for `node001` that is missing the module in the software image, the module could be configured to run on the node from the software image as follows:

  Example

  ```
  [root@basecm11 ~]# ssh node001 "lsmod | grep 8021q"
  [root@basecm11 ~]# cmsh
  [basecm11]% softwareimage
  [basecm11]->softwareimage% use default-image
  [basecm11->softwareimage[default-image]]% kernelmodules
  [basecm11->softwareimage[default-image]->kernelmodules]% list | grep 8021q
  [basecm11->softwareimage[default-image]->kernelmodules]% add 8021q
  [basecm11->softwareimage*[default-image*]->kernelmodules*[8021q*]]% commit
  ```

```
[basecm11->softwareimage[default-image]->kernelmodules[8021q]]% device use node001
[basecm11->device[node001]]% reboot
...
    some time after boot
[root@basecm11 ~]# ssh node001 "lsmod | grep 8021q"
8021q                  40960  0
garp                   16384  1 8021q
mrp                    20480  1 8021q
```

Rebooted nodes that use the modified software image then have the VLAN module available in the running kernel.

- The BIOS of the node must have the VLANID value set within the BIOS network options. If the BIOS does not support this setting, then PXE over VLAN cannot work. For a NIC that is missing this in the BIOS, the NIC hardware provider may sometimes have a BIOS update that supports this setting.
- The VLANID value should be set in the kernel parameters. Kernel parameters for a node can be specified in `cmsh` with the `kernelparameters` setting of `softwareimage` mode (section 9.3.4). An example where the VLANID is appended to some existing parameters could be:

Example

```
[root@basecm11 ~]# cmsh
[basecm11]% softwareimage use default-image
[basecm11->softwareimage[default-image]]% append kernelparameters " VLANID=89"
[basecm11->softwareimage*[default-image*]]% commit
```

5.4 Node-Installer

After the kernel has started up, and the ramdisk kernel modules are in place on the node, the node launches the node-installer.

The node-installer is a software image (section 9.4.4) provided by the head node. It interacts with CMDaemon on the head node and takes care of the rest of the boot process.

As an aside, the node-installer modifies some files (Appendix A.2.3) on the node it is installing to, so that they differ from the otherwise-expected pre-init stage Linux system. Such modifications can be prevented by a `frozenFilesPerNode` or `frozenFilesPerCategory` directive, as documented within the `node-installer.conf` file, and explained in greater detail on page 921.

Once the node-installer has completed its tasks, the local drive of the node has a complete Linux pre-init stage system. The node-installer ends by calling `/sbin/init` from the local drive and the boot process then proceeds as a normal Linux boot.

The steps the node-installer goes through for each node are:

1. requesting a node certificate (section 5.4.1)
2. deciding or selecting node configuration (section 5.4.2)
3. starting up all network interfaces (section 5.4.3)
4. determining install-mode type and execution mode (section 5.4.4)
5. running `initialize` scripts (section 5.4.5)
6. checking partitions, mounting filesystems (section 5.4.6)
7. synchronizing the local drive with the correct software image (section 5.4.7)

8. writing network configuration files to the local drive (section 5.4.8)
9. creating an `/etc/fstab` file on the local drive (section 5.4.9)
10. installing GRUB bootloader if configured by BCM (section 5.4.10), and initializing SELinux if it has been installed and configured (Chapter 12 of the *Installation Manual*)
11. running `finalize` scripts (section 5.4.11)
12. unloading specific drivers no longer needed (section 5.4.12)
13. switching the root device to the local drive and calling `/sbin/init` (section 5.4.13)

These 13 node-installer steps and related matters are described in detail in the corresponding sections 5.4.1–5.4.13.

5.4.1 Requesting A Node Certificate

Each node communicates with the CMDaemon on the head node using a certificate. If no certificate is found, it automatically requests one from CMDaemon running on the head node (figure 5.8).

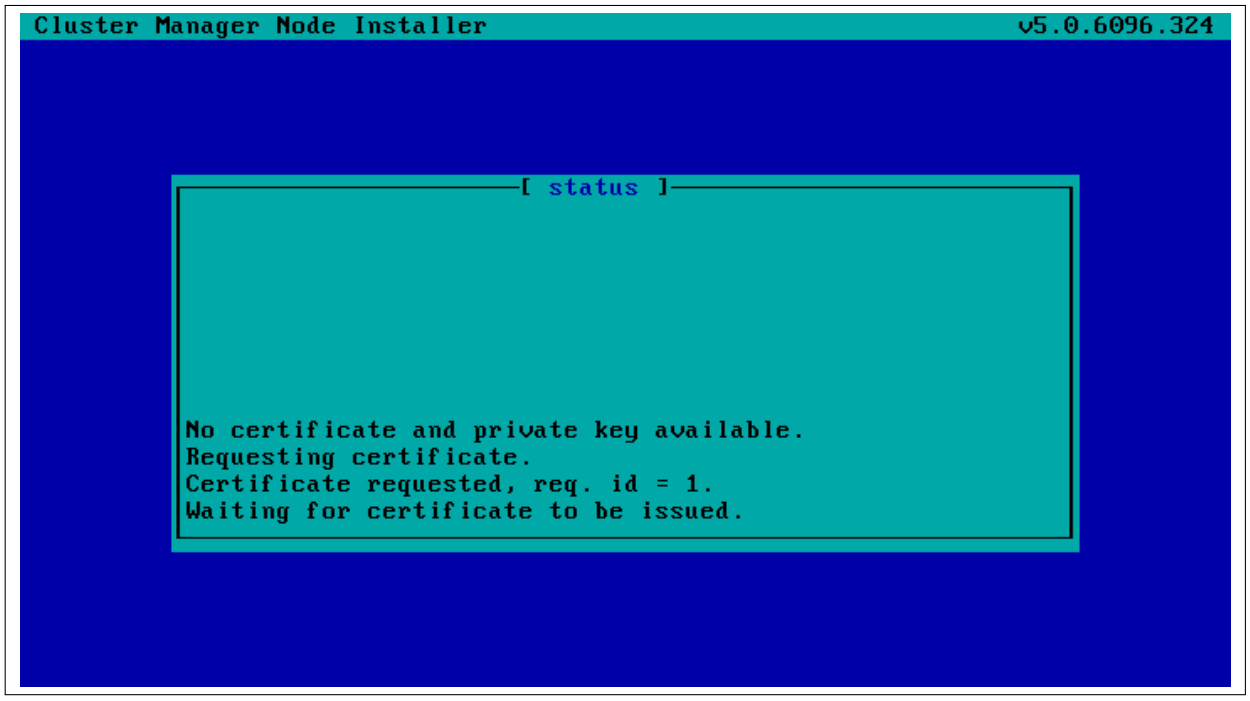


Figure 5.8: Certificate Request

The certificate is stored on the head node in `/cm/node-installer/certificates/` by MAC address.

Certificate Auto-signing

Certificate auto-signing means the cluster management daemon automatically signs a certificate signing request (CSR) that has been requested by a node. Certificate auto-signing can be configured from within `partition` mode of `cmsh`, with the `signinstallercertificates` parameter. It can take one of the following values:

- `AUTO` (the default)
- `MANUAL`

Example

```
[root@basecm11 ~]# cmsh
[basecm11]% partition
[basecm11->partition[base]]% set signinstallercertificates auto
```

For Base View, the navigation path is:

`Cluster > Settings > Sign installer certificates`

For untrusted networks, it may be wiser to approve certificate requests manually to prevent new nodes being added automatically without getting noticed.

Disabling certificate auto-signing for all networks can be done by setting `signinstallercertificates` to `MANUAL`.

Instead of disabling certificate autosigning for all networks, a finer tuning can be carried out for individual networks. This requires that `signinstallercertificates` be set to `AUTO` in `partition` mode. The `allowautosign` parameter in `network` mode can then be set for a particular network, and it can take one of the following values:

- `Always`
- `Automatic` (the default)
- `Never`
- `Secret`

Example

```
[root@basecm11 ~]# cmsh
[basecm11]% network use internalnet
[basecm11->network[internalnet]]% set allowautosign automatic <TAB><TAB>
always     automatic  never      secret
```

If `Always` is set, then incoming CSRs from all types of networks are automatically auto-signed.

If `Automatic` is set, then only networks that are of type internal are automatically auto-signed.

If `Never` is set, then all incoming CSRs that come in for that network have to be manually approved.

The value `Secret` is required for the `globalnet` network, for edge sites. A node on an edge site uses a shared secret that is passed along with the node request. The secret is set during edge site setup by BCM (section 2.1.1 of the *Edge Manual*).

Manual Approval Of A CSR

Approval of the CSR from a regular node (not an edge node): Manual approval of a CSR is typically done from within `certs` mode. A list of requests can be found, and from the list, the appropriate unsigned request can be signed and issued. The following session illustrates the process:

```
[basecm11->cert]% listrequests
Request ID   Client type  Session ID   Autosign Name
------------ ------------ ------------ -------- ------------------
6            installer    42949672986  No       fa-16-3e-22-cd-13
[basecm11->cert]% issuecertificate 6
Issued 6
```

Approval of the CSR from an edge node: If the shared secret has not been set for the edge director—that is, if it has not been stored locally on the edge director, or if it has not been passed on using the installation medium—then the node-installer prompts for the secret the first time that it boots. If the secret that is typed in matches the site secret, then a CSR from the edge director is handled by the head node, and a signed certificate is issued.

The edge compute nodes pick up their secret from the director. If the director does not have the secret, then the compute node's node-installer prompts for the secret on first boot. Once the secret is

set, then edge compute node sends its CSR to the head node (using the edge director) and gets a signed certificate automatically.

Section 2.3 has more information on certificate management in general.

Certificate Storage And Removal Implications

After receiving a valid certificate, the node-installer stores it in `/cm/node-installer/certificates/<`*node mac address*`>/` on the head node. This directory is NFS exported to the nodes, but can only be accessed by the `root` user. The node-installer does not request a new certificate if it finds a certificate in this directory, valid or invalid.

If an invalid certificate is received, the screen displays a communication error. Removing the node's corresponding certificate directory allows the node-installer to request a new certificate and proceed further.

5.4.2 Deciding Or Selecting Node Configuration

Once communication with the head node CMDaemon is established, the node-installer tries to identify the node it is running on so that it can select a configuration from CMDaemon's record for it, if any such record exists. It correlates any node configuration the node is expected to have according to network hardware detected. If there are issues during this correlation process then the administrator is prompted to select a node configuration until all nodes finally have a configuration.

Possible Node Configuration Scenarios

The correlations process and corresponding scenarios are now covered in more detail:

It starts with the node-installer sending a query to CMDaemon to check if the MAC address used for net booting the node is already associated with a node in the records of CMDaemon. In particular, it checks the MAC address for a match against the existing *node configuration* properties, and decides whether the node is *known* or *new*.

- the node is **known** if the query matches a node configuration. It means that node has been booted before.
- the node is **new** if no configuration is found.

In both cases the node-installer then asks CMDaemon to find out if the node is connected to an Ethernet switch, and if so, to which port. Setting up Ethernet switches for port detection is covered in section 3.8.

If a port is detected for the node, the node-installer queries CMDaemon for a node configuration associated with the detected Ethernet switch port. If a port is not detected for the node, then either the hardware involved with port detection needs checking, or a node configuration must be selected manually.

There are thus several scenarios:

1. The node is new, and an Ethernet switch port is detected. A node configuration associated with the port is found. The node-installer suggests to the administrator that the new node should use this configuration, and displays the configuration along with a confirmation dialog (figure 5.9). This suggestion can be interrupted, and other node configurations can be selected manually instead through a sub-dialog (figure 5.10). By default (in the main dialog), the original suggestion is accepted after a timeout.

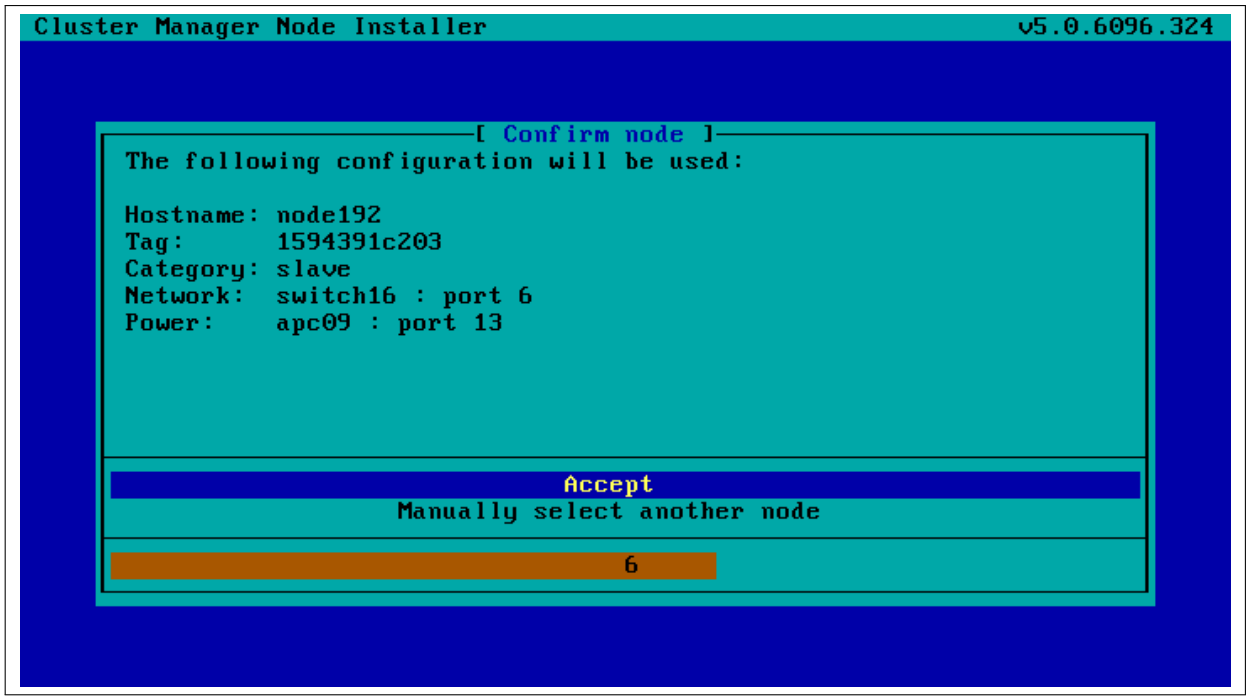


Figure 5.9: Scenarios: Configuration Found, Confirm Node Configuration

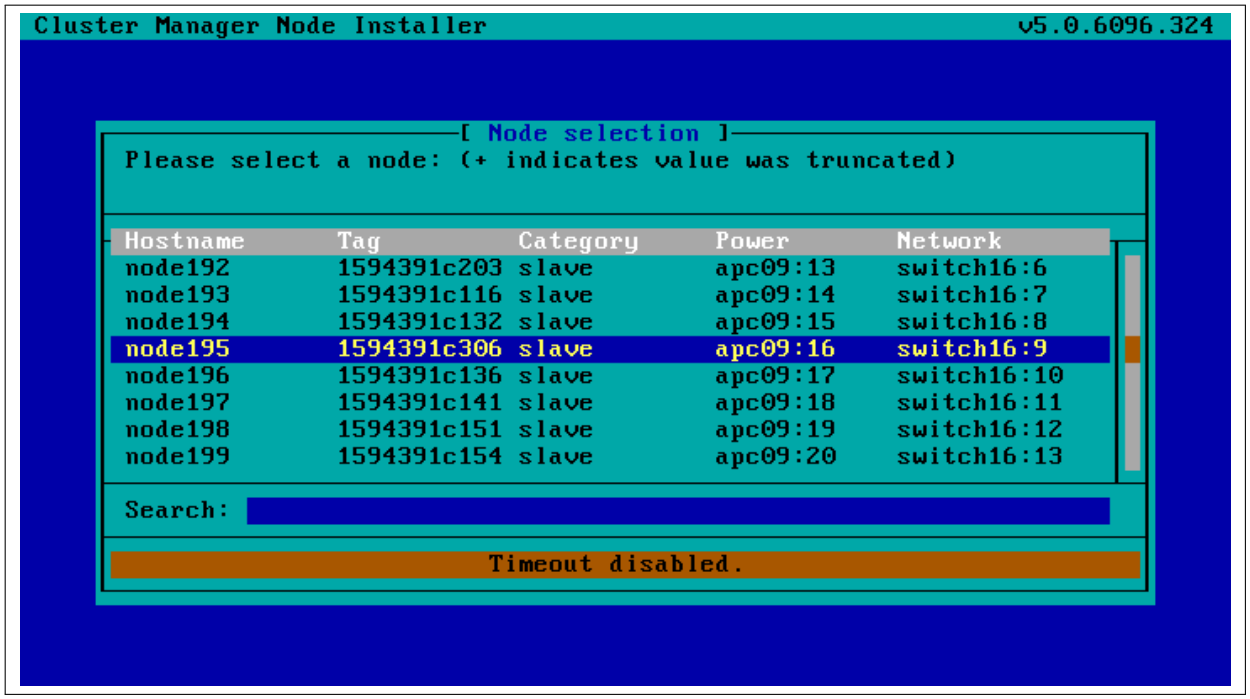


Figure 5.10: Scenarios: Node Selection Sub-Dialog

2. The node is new, and an Ethernet switch port is detected. A node configuration associated with the port is not found. The node-installer then displays a dialog that allows the administrator to either retry Ethernet switch port detection (figure 5.11) or to drop into a sub-dialog to manually select a node configuration (figure 5.10). By default, port detection is retried after a timeout.

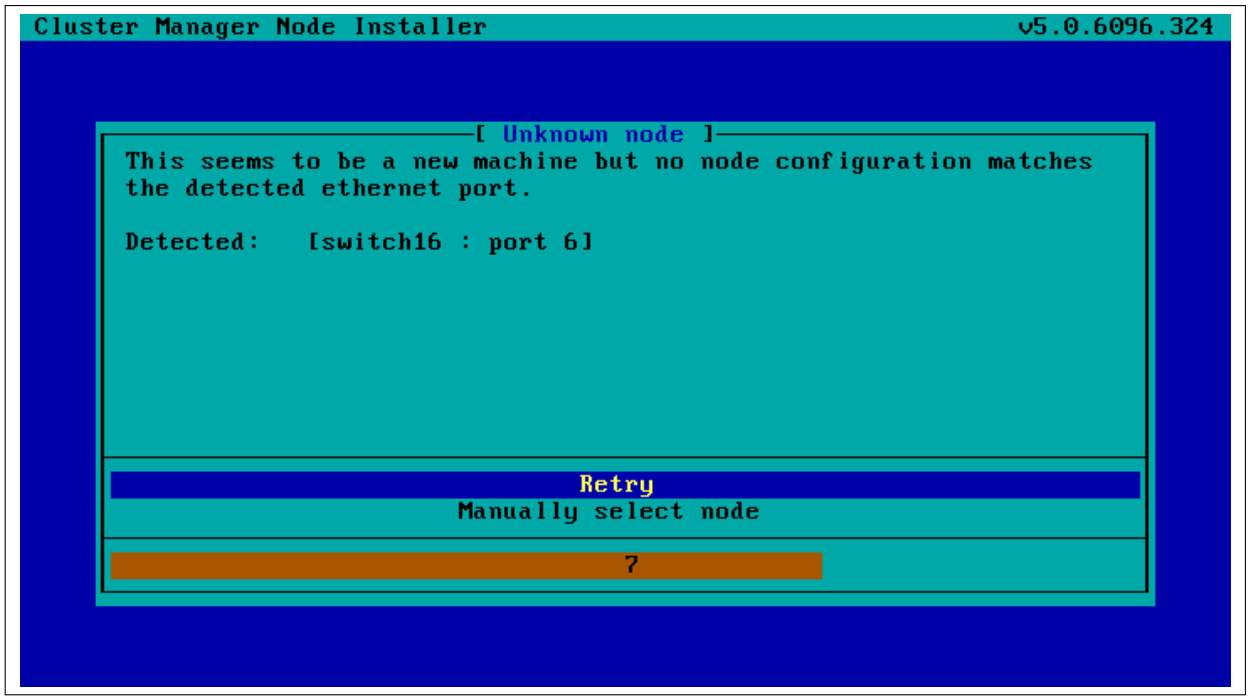


Figure 5.11: Scenarios: Unknown Node, Ethernet Port Detected

3. The node is new, and an Ethernet switch port is not detected. The node-installer then displays a dialog that allows the user to either retry Ethernet switch port detection (figure 5.12) or to drop into a sub-dialog to manually select a node configuration (figure 5.10). By default, port detection is retried after a timeout.

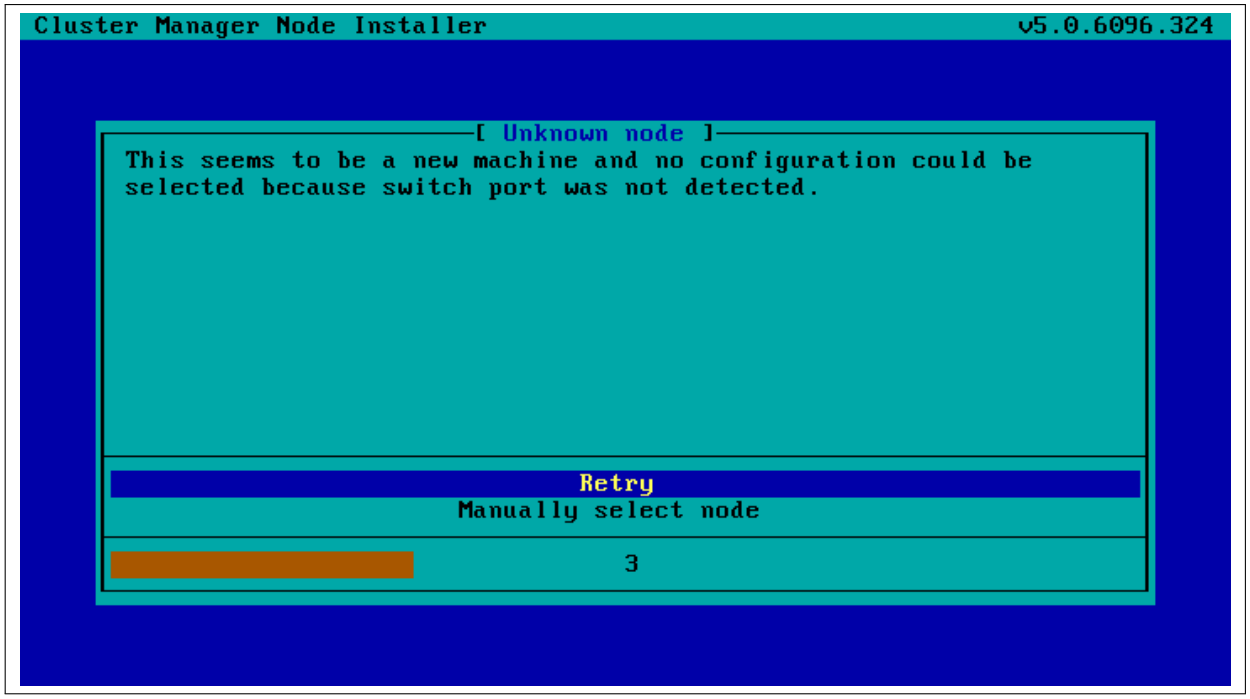


Figure 5.12: Scenarios: Unknown Node, No Ethernet Port Detected

4. The node is known, and an Ethernet switch port is detected. The configuration associated with the

port is the same as the configuration associated with the node's MAC address. The node-installer then displays the configuration as a suggestion along with a confirmation dialog (figure 5.9). The suggestion can be interrupted, and other node configurations can be selected manually instead through a sub-dialog (figure 5.10). By default (in the main dialog), the original suggestion is accepted after a timeout.

5. The node is known, and an Ethernet switch port is detected. However, the configuration associated with the port is not the same as the configuration associated with the node's MAC address. This is called a *port mismatch*. This type of port mismatch situation occurs typically during a mistaken *node swap*, when two nodes are taken out of the cluster and returned, but their positions are swapped by mistake (or equivalently, they are returned to the correct place in the cluster, but the switch ports they connect to are swapped by mistake). To prevent configuration mistakes, the node-installer displays a port mismatch dialog (figure 5.13) allowing the user to retry, accept a node configuration that is associated with the detected Ethernet port, or to manually select another node configuration using a sub-dialog (figure 5.10). By default (in the main port mismatch dialog), port detection is retried after a timeout.

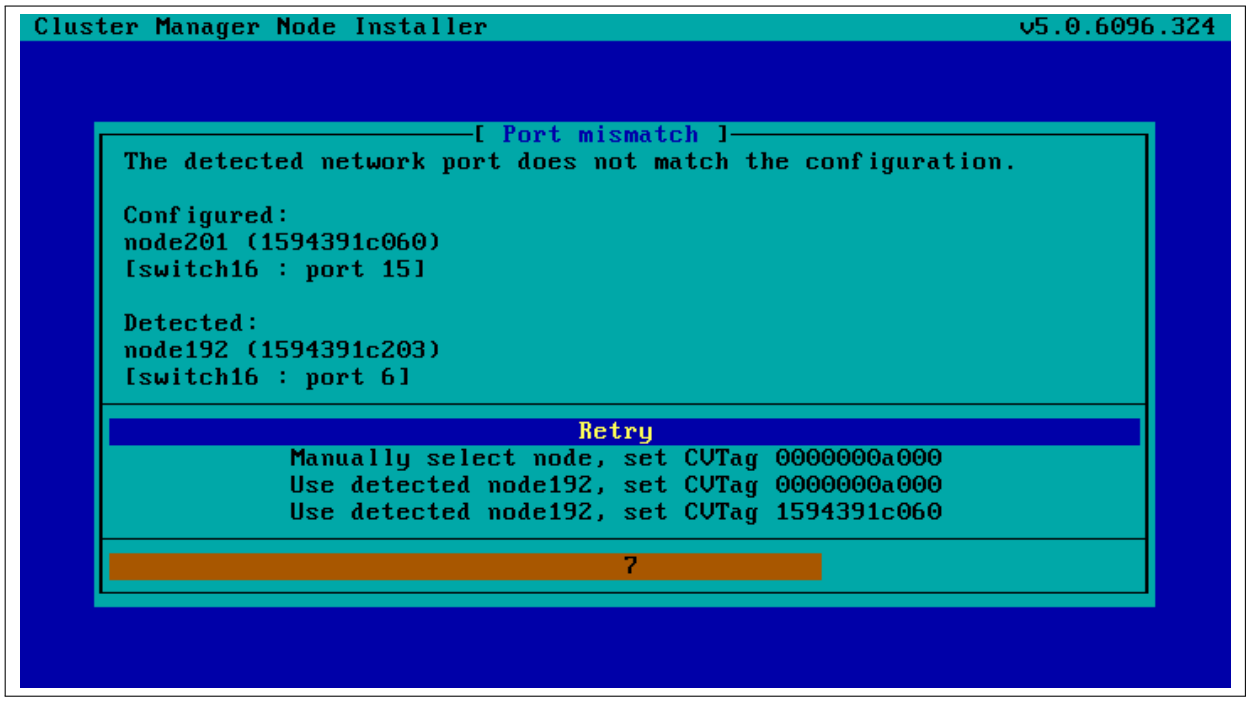


Figure 5.13: Scenarios: Port Mismatch Dialog

6. The node is known, and an Ethernet switch port is not detected. However, the configuration associated with the node's MAC address does have an Ethernet port associated with it. This is also considered a port mismatch. To prevent configuration mistakes, the node-installer displays a port mismatch dialog similar to figure 5.13, allowing the user to retry or to drop into a sub-dialog and manually select a node configuration that may work.

   However, a more likely solution in most cases is to:

   - either clear the switch port configuration in the cluster manager so that switch port detection is not attempted. For example, for `node001`, this can be done by running this `cmsh` command on the head node:
     ```
     cmsh -c "device clear node001 switchports; commit"
     ```
   - or enable switch port detection on the switch. This is usually quite straightforward, but may

require going through the manuals or software application that the switch manufacturer has provided.

By default (in the port mismatch dialog), port detection is retried after a timeout. This means that if the administrator clears the switch port configuration or enables switch port detection, the node-installer is able to continue automatically with a consistent configuration.

7. The node is known, and an Ethernet switch port is detected. However, the configuration associated with the node's MAC address has no Ethernet switch port associated with it. This is not considered a port mismatch but an unset switch port configuration, and it typically occurs if switch port configuration has not been carried out, whether by mistake or deliberately. The node-installer displays the configuration as a suggestion along with a confirmation dialog (figure 5.14). The suggestion can be interrupted, and other node configurations can be selected manually instead using a sub-dialog. By default (in the main dialog) the configuration is accepted after a timeout.

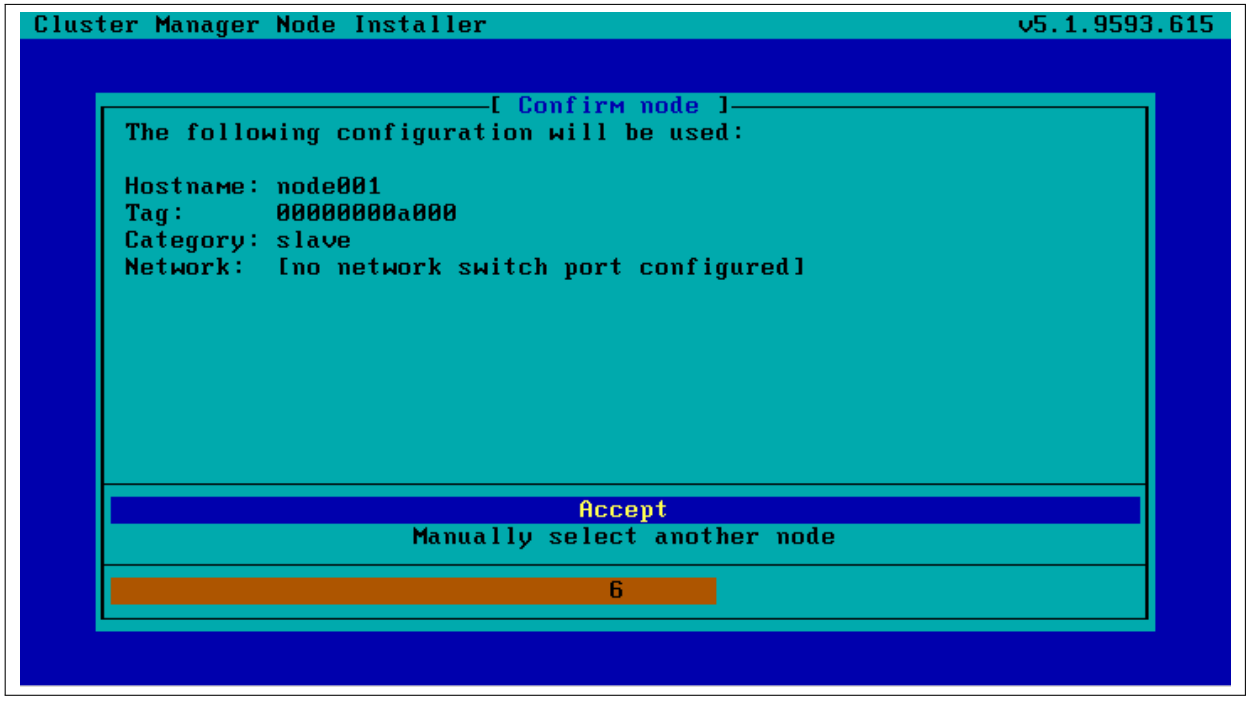


Figure 5.14: Scenarios: Port Unset Dialog

A truth table summarizing the scenarios is helpful:

Scenario	Node known?	Switch port detected?	Switch port configuration found?	Switch port configuration conflicts with node configuration?
1	No	Yes	Yes	No
2	No	Yes	No	No
3	No	No	No	No
4	Yes	Yes	Yes	No
5	Yes	Yes	Yes	Yes (configurations differ)
6	Yes	No	Yes	Yes (port expected by MAC configuration not found)
7	Yes	Yes	No	No (port not expected by MAC configuration)

In these scenarios, whenever the user manually selects a node configuration in the prompt dialog, an attempt to detect an Ethernet switch port is repeated. If a port mismatch still occurs, it is handled by the system as if the user has not made a selection.

Summary Of Behavior During Hardware Changes

The logic of the scenarios means that an unpreconfigured node always boots to a dialog loop requiring manual intervention during a first install (scenarios 2 and 3). For subsequent boots the behavior is:

- If the node MAC hardware has changed (scenarios 1, 2, 3):
  - if the node is new and the detected port has a configuration, the node automatically boots to that configuration (scenario 1).
  - else manual intervention is needed (scenarios 2, 3)
- If the node MAC hardware has not changed (scenarios 4, 5, 6, 7):
  - if there is no port mismatch, the node automatically boots to its last configuration (scenarios 4, 7).
  - else manual intervention is needed (scenarios 5, 6).

The `newnodes` Command

`newnodes` **basic use:** New nodes that have not been configured yet can be detected using the `newnodes` command from within the `device` mode of `cmsh`. A new node is detected when it reaches the node-installer stage after booting, and contacts the head node.

Example

```
[basecm11->device]% newnodes
The following nodes (in order of appearance) are waiting to be assigned:
MAC               First appeared                Detected on switch port
----------------  ----------------------------  -----------------------
00:0C:29:01:0F:F8  Mon, 14 Feb 2011 10:16:00 CET  [no port detected]
```

At this point the node-installer is seen by the administrator to be looping, waiting for input on what node name is to be assigned to the new node.

The nodes can be uniquely identified by their MAC address or switch port address.

The port and switch to which a particular MAC address is connected can be discovered by using the `showport` command (section 3.9.4). After confirming that they are appropriate, the `switchports` property for the specified device can be `set` to the port and switch values.

Example

```
[basecm11->device]% showport 00:0C:29:01:0F:F8
switch01:8
[basecm11->device]% set node003 switchports switch01:8
[basecm11->device*]% commit
```

When the node name (`node003` in the preceding example) is assigned, the node-installer stops looping and goes ahead with the installation to the node.

The preceding basic use of `newnodes` is useful for small numbers of nodes. For larger number of nodes, the advanced options of `newnodes` may help carry out node-to-MAC assignment with less effort.

`newnodes` **advanced use—options:** The list of MAC addresses discovered by a `newnodes` command can be assigned in various ways to nodes specified by the administrator. Node objects should be created in advance to allow the assignment to take place. The easiest way to set up node objects in `cmsh` is to use the `--clone` option of the `foreach` command (section 2.5.5, page 73).

The advanced options of `newnodes` are particularly useful for quickly assigning node names to specific physical nodes. All that is needed is to power the nodes up in the right order. For nodes with the same hardware, the node that is powered up first reaches the stage where it tries to connect with the node-installer first. So its MAC address is detected first, and arrives on the list generated by `newnodes` first. If some time after the first node is powered up, the second node is powered up, then its MAC address becomes the second MAC address on the list, and so on for the third, fourth, and further nodes.

When assigning node names to a physical node, on a cluster that has no such assignment already, the first node that arrived on the list gets assigned the name `node001`, the second node that arrived on the list gets assigned the name `node002` and so on.

The advanced options are shown in `device` mode by running the `help newnodes` command. The options can be introduced as being of three kinds: straightforward, grouping, and miscellaneous:

- The straightforward options:

  `-n|--nodes`

  `-w|--write`

  `-s|--save`

  Usually the most straightforward way to assign the nodes is to use the `-n` option, which accepts a list of nodes, together with a `-w` or `-s` option. The `-w` (`--write`) option sets the order of nodes to the corresponding order of listed MAC addresses, and is the same as setting an object in `cmsh`. The `-s` (`--save`) option is the same as setting and committing an object in `cmsh`, so `-s` implies a `-w` option is run at the same time.

  So, for example, if 8 new nodes are discovered by the node-installer on a cluster with no nodes so far, then:

  Example

  ```
  [basecm11->device]% newnodes -w -n node001..node008
  ```

  assigns (but does not commit) the sequence node001 to node008 the new MAC address according to the sequence of MAC addresses displaying on the list.

- The grouping options:

  `-g|--group`

  `-c|--category`

  `-h|--chassis`

  `-r|--rack`

  The "`help newnodes`" command in `device` mode shows assignment options other than `-n` for a node range are possible. For example, the assignments can also be made for a group (`-g`), per category (`-c`), per chassis (`-h`), and per rack (`-r`).

- The miscellaneous options:

  `-f|--force`

  `-o|--offset`

By default, the `newnodes` command fails when it attempts to set a node name that is already taken. The `-f` (`--force`) option forces the new MAC address to be associated with the old node name. When used with an assignment grouping, (node range, group, category, chassis, or rack) all the nodes in the grouping lose their node-to-MAC assignments and get new assignments. The `-f` option should therefore be used with care.

The `-o` (`--offset`) option takes a number <*number*> and skips <*number*> nodes in the list of detected unknown nodes, before setting or saving values from the assignment grouping.

Examples of how to use the advanced options follow.

`newnodes` **advanced use—range assignment behavior example:** For example, supposing there is a cluster with nodes assigned all the way up to node022. That is, CMDaemon knows what node is assigned to what MAC address. For the discussion that follows, the three nodes node020, node021, node022 can be imagined as being physically in a rack of their own. This is simply to help to visualize a layout in the discussion and tables that follow and has no other significance. An additional 3 new, that is unassigned, nodes are placed in the rack, and allowed to boot and get to the node-installer stage.

The `newnodes` command discovers the new MAC addresses of the new nodes when they reach their node-installer stage, as before (the switch port column is omitted in the following text for convenience):

Example

```
[basecm11->device]% newnodes
MAC                First appeared
-----------------  ----------------------------
00:0C:29:EF:40:2A  Tue, 01 Nov 2011 11:42:31 CET
00:0C:29:95:D3:5B  Tue, 01 Nov 2011 11:46:25 CET
00:0C:29:65:9A:3C  Tue, 01 Nov 2011 11:47:13 CET
```

The assignment of MAC to node address could be carried out as follows:

Example

```
[basecm11->device]% newnodes -s -n node023..node025
MAC                 First appeared                Hostname
------------------- ----------------------------  --------
00:0C:29:EF:40:2A   Tue, 01 Nov 2011 11:42:31 CET  node023
00:0C:29:95:D3:5B   Tue, 01 Nov 2011 11:46:25 CET  node024
00:0C:29:65:9A:3C   Tue, 01 Nov 2011 11:47:13 CET  node025
```

Once this is done, the node-installer is able to stop looping, and to go ahead and install the new nodes with an image.

The physical layout in the rack may then look as indicated by this:

before	after	MAC
node020	node020	
node021	node021	
node022	node022	
	node023	...A
	node024	...B
	node025	...C

Here, `node023` is the node with the MAC address ending in `A`.

If instead of the previous `newnodes` command, an offset of 1 is used to skip assigning the first new node:

Example

```
[basecm11->device]% newnodes -s -o 1 node024..node025
```

then the rack layout looks like:

before	after	MAC
node020	node020	
node021	node021	
node022	node022	
	unassigned	...A
	node024	...B
	node025	...C

Here, *unassigned* is where `node023` of the previous example is physically located, that is, the node with the MAC address `...A`. The lack of assignment means there is actually no association of the name `node023` with that MAC address, due to the `newnodes` command having skipped over it with the `-o` option.

If instead the assignment is done with:

Example

```
[basecm11->device]% newnodes -s 1 node024..node026
```

then the `node023` name is unassigned, and the name `node024` is assigned instead to the node with the MAC address `...A`, so that the rack layout looks like:

before	after	MAC
node020	node020	
node021	node021	
node022	node022	
	node024	...A
	node025	...B
	node026	...C

`newnodes` **advanced use—assignment grouping example:** Node range assignments are one way of using `newnodes`. However assignments can also be made to a category, a rack, or a chassis. For example, with Base View assigning node names to a rack can be done from the `Racks` option of the node. For example, to add a `node001` to a `rack1`, the navigation path would be:

`Devices > All Devices > node001`  `> Settings > Rack + > rack1`

In `cmsh`, the assignment of multiple node names to a rack can conveniently be done with a `foreach` loop from within `device` mode:

Example

```
[basecm11->device]% foreach -n node020..node029 (set rack rack02)
[basecm11->device*]% commit
[basecm11->device]% foreach -n node030..node039 (set rack rack03)
[basecm11->device*]% commit
```

The assignment of node names with the physical node in the rack can then be arranged as follows: If the nodes are identical hardware, and are powered up in numerical sequence, from node020 to node039, with a few seconds in between, then the list that the basic newnodes command (without options) displays is arranged in the same numerical sequence. Assigning the list in the rack order can then be done by running:

Example

```
[basecm11->device]% newnodes -s -r rack02..rack03
```

If it turns out that the boot order was done very randomly and incorrectly for all of rack02, and that the assignment for rack02 needs to be done again, then a simple way to deal with it is to bring down the nodes of rack02, then clear out all of the rack02 current MAC associations, and redo them according to the correct boot order:

Example

```
[basecm11->device]% foreach -r rack02 ( clear mac ) ; commit
 ...removes MAC association with nodes from CMDaemon...

 ...now reboot nodes in rack02 in sequence (not with BCM)...

[basecm11->device]% newnodes

 ...shows sequence as the nodes come up...

[basecm11->device]% newnodes -s -r rack02

 ...assigns sequence in boot order...
```

newnodes **advanced use—assignment forcing example:** The --force option can be used in the following case: Supposing that node022 fails, and a new node hardware comes in to replace it. The new regular node has a new MAC address. So, as explained by scenario 3 (section 5.4.2), if there is no switch port assignment in operation for the nodes, then the node-installer loops around, waiting for intervention. (With switch port assignment in place, scenario 1 means the new node simply boots up by default and becomes the new node022 without further intervention.)

This situation can be dealt with from the command line by:

- accepting the node configuration at the regular node console, using a sub-dialog
- accepting the node configuration using cmsh, without needing to be at the regular node console:

  ```
  [basecm11->device]% newnodes -s -f -n node022
  ```

Device Identification

The *Device Identification* resource can be accessed using the Base View navigation path:

Devices > Device Identification

Device Identification is the Base View version of the newnodes command of cmsh. It lists the unassigned new nodes that the head node detects. These are displayed in the pane along with their MAC addresses and switch ports (figure 5.15).

Similarly to newnodes in cmsh, the Base View pane lists the MAC address of any unassigned node that the head node detects, and shows the associated detected switch port for the node. Also, just as

for `newnodes`, the Base View pane can help assign a node name to the node, assuming the node object exists. After assignment is done, the new status should be saved.

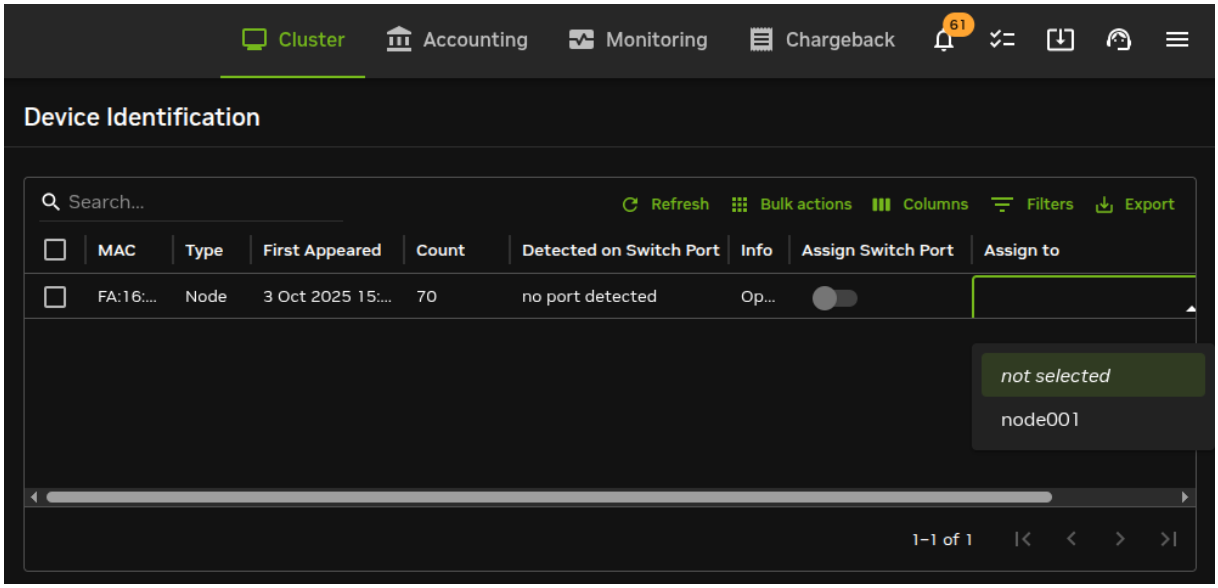


Figure 5.15: Device Identification Pane: Node Assignment

The most useful way of using the device identification resource is for node assignment in large clusters.

To do this, it is assumed that the node objects have already been created for the new nodes. The creation of the node objects means that the node names exist, and so assignment to the node names is able to take place. An easy way to create many nodes in Base View, set their provisioning interface, and set their IP addresses is described in the section on the *node creation wizard* (section 5.7.2). Node objects can also be created easily in large numbers by using `cmsh`'s `foreach` loop command on a node with the `--clone` option (section 2.5.5, page 73).

The nodes are also assumed to be set for net booting, typically set from a BIOS setting.

The physical nodes are then powered up in an arranged order. Because they are unknown new nodes, the node-installer keeps looping after a timeout. The head node in the meantime detects the new MAC addresses and switch ports in the sequence in which they first have come up and lists them in that order.

By default, all these newly detected nodes are set to an install mode of `auto` (section 5.4.4), which means that their numbering goes up sequentially from whatever number is assigned to the preceding node in the list. Thus, if there are 10 new unassigned nodes that are brought into the cluster, and the first node in the list is assigned to the first available number, say `node327`, then the remaining nodes can automatically be assigned in sequence as nodes `node328` to `node336`.

- With `cmsh` this can be done with the `newnodes -s` option, as described on page 271.
- With Base View this can be done in the `Devices` > `Device Identification` pane, by:
  - clicking on the checkbox that selects all the rows in the list of new nodes, then
  - clicking on the `Bulk actions > Autofill devices` option to automatically fill the nodes, then
  - clicking on the `Bulk actions > Commit` option to commit the changes

After the assignment, the node-installer looping process on the new nodes notices that the nodes are now known, so it breaks out of the loop, and installation goes ahead without any intervention needed at the node console.

5.4.3 Starting Up All Network Interfaces

At the end of section 5.4.2, the node-installer knows which node it is running on, and has decided what its node configuration is.

Starting Up All Provisioning Network Interfaces

It now gets on with setting up the IP addresses on the provisioning interfaces required for the node-installer, while taking care of matters that come up on the way:

Avoiding duplicate IP addresses: The node-installer brings up all the network interfaces configured for the node. Before starting each interface, the node-installer first checks if the IP address that is about to be used is not already in use by another device. If it is, then a warning and retry dialog is displayed until the IP address conflict is resolved.

Using `BOOTIF` to specify the boot interface: `BOOTIF` is a special name for one of the possible interfaces. The node-installer automatically translates `BOOTIF` into the name of the device, such as `eth0` or `eth1`, used for network booting. This is useful for a machine with multiple network interfaces where it can be unclear whether to specify, for example, `eth0` or `eth1` for the interface that was used for booting. Using the name `BOOTIF` instead means that the underlying device, `eth0` or `eth1` in this example, does not need to be specified in the first place.

Halting on missing kernel modules for the interface: For some interface types like VLAN and channel bonding, the node-installer halts if the required kernel modules are not loaded or are loaded with the wrong module options. In this case the kernel modules configuration for the relevant software image should be reviewed. Recreating the ramdisk and rebooting the node to get the interfaces up again may be necessary, as described in section 5.8.6.

Bringing Up Non-Provisioning Network Interfaces

Provisioning interfaces are by default automatically brought up during the init stage, as the node is fully booted up. The BMC and non-provisioning interfaces on the other hand have a different behavior:

Bringing Up And Initializing BMC Interfaces: If a BMC interface is present and powered up, then it is expected to be running at least with layer 2 activity (ethernet). It can be initialized in the node configuration (section 3.7) with an IP address, netmask and user/password settings so that layer 3 (TCP/IP) networking works for it. BMC networking runs independently of node networking.

Bringing up non-BMC, non-provisioning network interfaces: Non-provisioning interfaces are inactive unless they are explicitly brought up. BCM can configure how these non-provisioning interfaces are brought up by using the `bringupduringinstall` parameter, which can take the following values:

- `yes`: Brings the interface up during the pre-init stage
- `no`: Keeps the interface down during the pre-init stage. This is the default for non-provisioning interfaces.
- `yesandkeep`: Brings the interface up during the pre-init stage, and keeps it up during the transition to the init stage.

Bringing Up And Keeping Up Provisioning Network Interfaces

The preceding `bringupduringinstall` parameter is not generally supported for provisioning interfaces. However the `yesandkeep` value does work for provisioning interfaces too, under some conditions:

- `yesandkeep`: Brings the interface up during the pre-init stage, and keeps it up during the transition to the init stage, for the following provisioning devices:

- Ethernet device interfaces using a leased DHCP address
- InfiniBand device interfaces running with distribution OFED stacks

Restarting The Network Interfaces

At the end of this step (i.e. section 5.4.3) the network interfaces are up. When the node-installer has completed the remainder of its 13 steps (sections 5.4.4–5.4.13), control is handed over to the local `init` process running on the local drive. During this handover, the node-installer brings down all network devices. These are then brought back up again by `init` by the distribution's standard networking `init` scripts, which run from the local drive and expect networking devices to be down to begin with.

5.4.4 Determining Install-mode Type And Execution Mode

Stored *install-mode* values decide whether synchronization is to be applied fully to the local drive of the node, only for some parts of its filesystem, not at all, or even whether to drop into a maintenance mode instead.

Related to install-mode values are execution mode values (page 278) that determine whether to apply the install-mode values to the next boot, to new nodes only, to individual nodes or to a category of nodes.

Related to execution mode values is the confirmation requirement toggle value (page 279) in case a full installation is to take place.

These values are merely determined at this stage; nothing is executed yet.

Install-mode Values

The install-mode can have one of five values: `AUTO`, `FULL`, `MAIN`, `NOSYNC`, and `SKIP`. It should be understood that the term "install-mode" implies that these values operate only during the node-installer phase.

NOTE: For example, `imageupdate` (section 5.6.2), which is run by CMDaemon, ignores these settings, which is as expected. This means that, for example, if `imageupdate` is run with `NOSYNC` set, then the head node image is still synchronized over as usual to the regular node while the node is up. It is only during node boot, during the installer stage, that setting NOSYNC prevents synchronization.

- If the install-mode is set to `FULL`, then the node-installer re-partitions, creates new filesystems and synchronizes a full image onto the local drive according a *partition layout*. This process wipes out all pre-boot drive content.

  A partition layout (Appendix D) includes defined values for the partitions, sizes, and filesystem types for the nodes being installed. An example of a partition layout is the default partition layout (Appendix D.3).

- If the install-mode is set to `AUTO`, then the node-installer checks the partition layout of the local drive against the node's stored configuration. If these do not match because, for example, the node is new, or if they are corrupted, then the node-installer recreates the partitions and filesystems by carrying out a `FULL` install. If however the drive partitions and filesystems are healthy, the node-installer only does an incremental software image synchronization. Synchronization tends to be quick because the software image and the local drive usually do not differ much.

  Synchronization also removes any extra local files that do not exist on the image, for the files and directories considered. Section 5.4.7 gives details on how it is decided what files and directories are considered.

- If the install-mode is set to `MAIN`, then the node-installer does not carry out a disk check, and goes on to maintenance mode, allowing manual investigation of specific problems. The local drive is untouched.

- If the install-mode is set to `NOSYNC`, and the partition layout check matches the stored XML configuration, then the node-installer skips synchronizing the image to the node, so that contents on the

local drive persist from the previous boot. An exception to this is the node certificate and key, that is the files /cm/local/apps/cmd/etc/cert.{pem|key}. These are updated from the head node if missing.

If however the partition layout does not match the stored configuration, a FULL image sync is triggered. Thus, for example, a burn session (Chapter 11 of the *Installation Manual*), with the default burn configuration which destroys the existing partition layout on a node, will trigger a FULL image sync on reboot after the burn session.

The NOSYNC setting should therefore not be regarded as a way to protect data. Ways to preserve data across node reboots are discussed in the section that discusses the FULL install confirmation settings (page 279).

NOSYNC is useful during mass planned node reboots when set with the nextinstallmode option of device mode. This sets the nodes to use the OS on the hard drive, during the next boot only, without an image sync:

Example

```
[basecm11]% device foreach -n node001..node999 (set nextinstallmode nosync)
[basecm11]% device commit
```

- If the install-mode is set to SKIP, then the node-installer does not carry out a check of the partitions and filesystems, and it also does not carry out a software image synchronization. If a node runs into problems with its drive content during a normal start up attempt, then this mode can perhaps be used to attempt data recovery on the node.

Install-mode Logging

The decision that is made is normally logged to the node-installer file, /var/log/node-installer on the head node.

Example

```
08:40:58 node001 node-installer: Installmode is: AUTO
08:40:58 node001 node-installer: Fetching disks setup.
08:40:58 node001 node-installer: Setting up environment for initialize scripts.
08:40:58 node001 node-installer: Initialize script for category default is empty.
08:40:59 node001 node-installer: Checking partitions and filesystems.
08:40:59 node001 node-installer: Updating device status: checking disks
08:40:59 node001 node-installer: Detecting device '/dev/sda': found
08:41:00 node001 node-installer: Number of partitions on sda is ok.
08:41:00 node001 node-installer: Size for /dev/sda1 is ok.
08:41:00 node001 node-installer: Checking if /dev/sda1 contains ext3 filesystem.
08:41:01 node001 node-installer: fsck.ext3 -a /dev/sda1
08:41:01 node001 node-installer: /dev/sda1: recovering journal
08:41:02 node001 node-installer: /dev/sda1: clean, 129522/1250928 files, 886932/5000000 blocks
08:41:02 node001 node-installer: Filesystem check on /dev/sda1 is ok.
08:41:02 node001 node-installer: Size for /dev/sda2 is wrong.
08:41:02 node001 node-installer: Partitions and/or filesystems are missing/corrupt. (Exit code\
 18, signal 0)
08:41:03 node001 node-installer: Creating new disk layout.
```

In this case the node-installer detects that the size of /dev/sda2 on the disk no longer matches the stored configuration, and triggers a full re-install. For further detail beyond that given by the node-installer log, the disks script at /cm/node-installer/scripts/disks on the head node can be examined. The node-installer checks the disk by calling the disks script. Exit codes, such as the 18 reported in the log example, are defined near the top of the disks script.

Install-mode's Execution Modes

Execution of an install-mode setting is possible in several ways, both permanently or just temporarily for the next boot. Execution can be set to apply to categories or individual nodes. The node-installer looks for install-mode execution settings in this order:

1. The "`New node installmode`" property of the node's category. This decides the install mode for a node that is detected to be new.

   It can be set for the `default` category using a Base View navigation path such as:

   `Grouping > Categories[default]` > `Settings > New node install mode`

   or using `cmsh` with a one-liner such as:

   ```
   cmsh -c "category use default; set newnodeinstallmode FULL; commit"
   ```

   By default, the "`New node installmode`" property is set to `FULL`.

2. The Install-mode setting as set by choosing a PXE menu option on the console of the node before it loads the kernel and ramdisk (figure 5.16). This only affects the current boot. By default the PXE menu install mode option is set to `AUTO`.

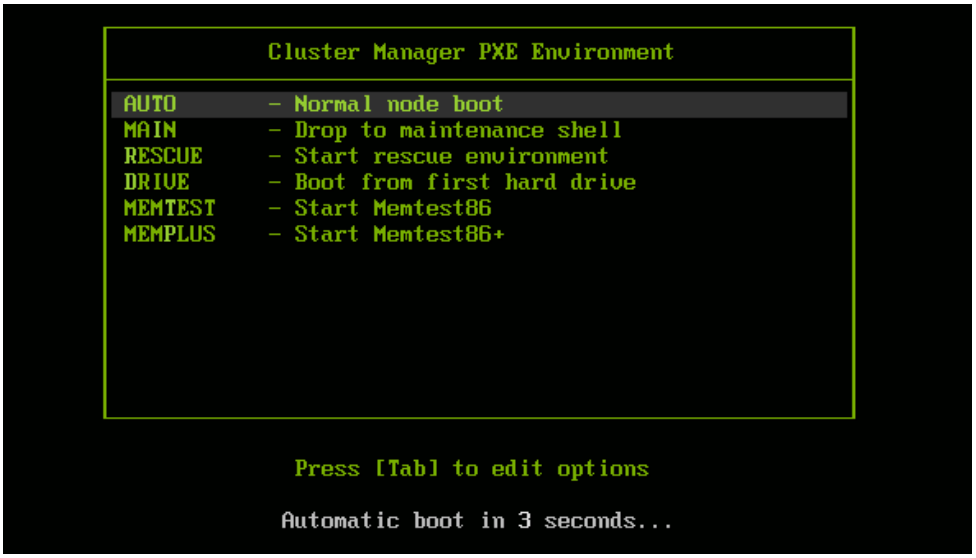


Figure 5.16: PXE Menu With Install-mode Set To `AUTO`

3. The "`Next boot install-mode`" property of the node configuration. This can be set for a node such as `node001` using the Base View navigation path:

   `Devices > All Devices > node001` > `Settings > Next install mode`

   It can also be set using `cmsh` with a one-liner:

   ```
   cmsh -c "device use node001; set nextinstallmode FULL; commit"
   ```

   The property is cleared when the node starts up again, after the node-installer finishes its installation tasks. So it is empty unless specifically set by the administrator during the current uptime for the node.

4. The `install-mode` property can be set in the node configuration using the Base View navigation path:

   `Devices > All Devices > node001` > `Settings > Install mode`

   or using `cmsh` with a one-liner such as:

```
cmsh -c "device use node001; set installmode FULL; commit"
```

By default, the install-mode property is auto-linked to the property set for `install-mode` for that category of node. Since the property for that node's category defaults to `AUTO`, the property for the `install-mode` of the node configuration defaults to "`AUTO (Category)`".

5. The `install-mode` property of the node's category. This can be set using Base View with a navigation path such as:
   `Grouping > Categories > [default]`  `> Settings > Install mode`
   or using `cmsh` with a one-liner such as:

   ```
   cmsh -c "category use default; set installmode FULL; commit"
   ```

   As already mentioned in point 4, the install-mode is set by default to `AUTO`.

6. A dialog on the console of the node (figure 5.17) gives the user a last opportunity to overrule the install-mode value as determined by the node-installer. By default, it is set to `AUTO`:

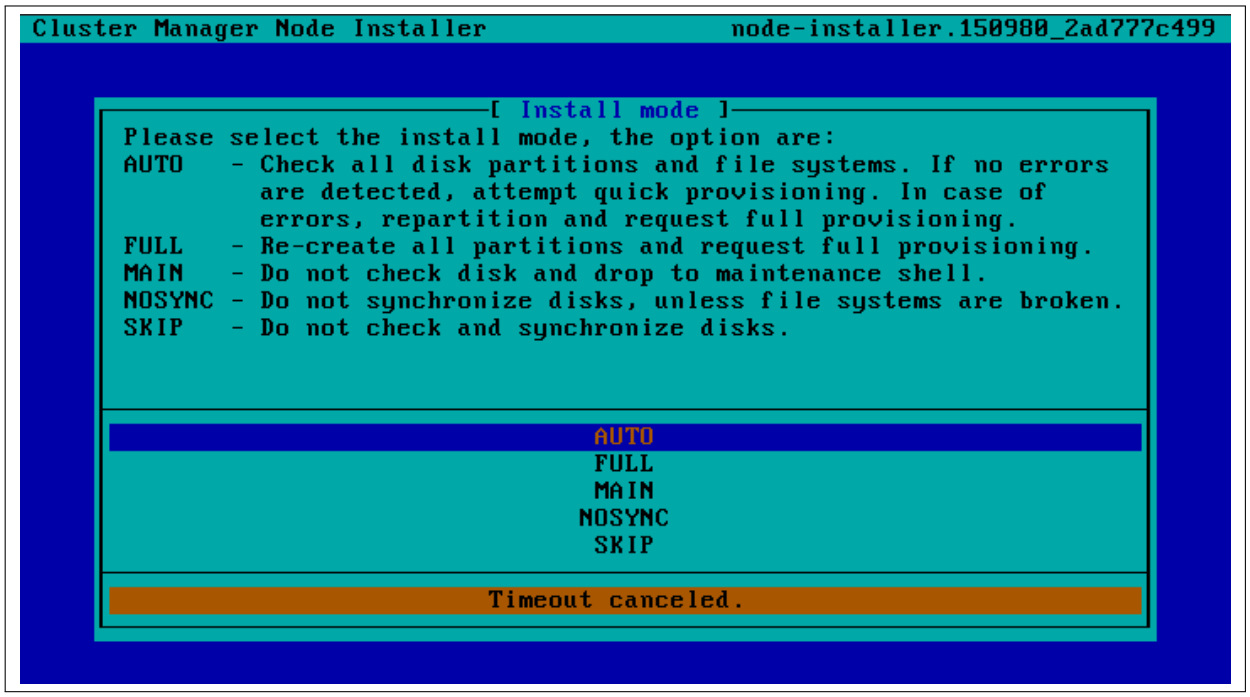


Figure 5.17: Install-mode Setting Option During Node-Installer Run

FULL Install Confirmation Using `datanode` Setting

Related to execution mode values is the ability to carry out a FULL install only after explicit confirmation, using the `datanode` property. This must be set in order to prompt for a confirmation, when a FULL installation is about to take place. If it is set, then the node-installer only goes ahead with the FULL install after the administrator has explicitly confirmed it.

The `datanode` property can be enabled in the node configuration of, for example, `node001` with Base View using the navigation path:

`Devices > All Devices > node001`  `> Settings > Data node`

Alternatively, the parameter `datanode` can be set using a `cmsh` one-liner as follows:

```
[root@basecm11 ~]# cmsh -c "device use node001; set datanode yes; commit"
```

The property can also be set at a category level. Since datanode is a boolean value, the actual value that is used used for a node is the result of the or operation for that value across the levels. The level at which a value works due to its boolean or non-boolean type is explained more on page 29.

Why the FULL install confirmation is useful: The reason for such a setting is that a FULL installation can be triggered by disk or partition changes, or by a change in the MAC address. If that happens, then:

- considering a drive, say, /dev/sda that fails, this means that any drive /dev/sdb would then normally become /dev/sda upon reboot. In that case an unwanted FULL install would not only be triggered by an install-mode settings of FULL, but also by the install-mode settings of AUTO or NOSYNC. Having the new, "accidental" /dev/sda have a FULL install is unlikely to be the intention, since it would probably contain useful data that the node-installer earlier left untouched.
- considering a node with a new MAC address, but with local storage containing useful data from earlier. In this case, too, an unwanted FULL install would not only be triggered by an install-mode setting of FULL, but also by the install-mode settings AUTO or NOSYNC.

Thus, in cases where nodes are used to store data, an explicit confirmation before overwriting local storage contents is a good idea. However, by default, no confirmation is asked for when a FULL installation is about to take place.

Carrying out the confirmation: When the confirmation is required, then it can be carried out by the administrator as follows:

- From the node console. A remote console launched from Base View or cmsh will also work if SOL connectivity has been configured.
- From cmsh, within device mode, using the installerinteractions command (some output elided):

  Example

```
[basecm11->device]% installerinteractions -w -n node001 --confirm
Hostname  Action
--------- ------------------------------------------
node001   Requesting FULL Install (partition mismatch)
[basecm11->device]%
...07:57:36 [notice] basecm11: node001 [ INSTALLER_CALLINGINIT ]...
[basecm11->device]%
...07:58:20 [notice] basecm11: node001 [   UP   ]
```

  The installerinteractions command then sets the node to a confirmed state. The other possible states are deny and pending.

  Besides confirmation, the installerinteractions command has options that include letting it:

  - deny the installation, and put it into maintenance mode
  - carry out a dry-run
  - carry out its actions for node groupings such as: node lists, node categories, node groups, chassis, racks, as are possible in the grouping options (page 72).

  Further details on the command can be viewed by running help installerinteractions.

An alternative way to avoid overwriting node storage: Besides the method of FULL install confirmation for `datanode`, there is a method based on XML assertions, that can also be used to prevent data loss on nodes.

It uses XML assertions to confirm that the physical drive is recognized (Appendix D.11).

A way to overwrite a specified block device: A related method is that sometimes, for reasons of performance or convenience, it may be desirable to clear data on particular block devices for a node, and carry it out during the next boot only. This can done by setting the block device names to be cleared as values to the parameter `Block devices cleared on next boot`. The values can be set in `cmsh` as follows:

```
[basecm11->device[node001]]% append blockdevicesclearedonnextboot /dev/sda /dev/sdb ; commit
```

The value of `blockdevicesclearedonnextboot` is automatically cleared after the node is rebooted. Clearing data in this way ignores any `datanode` or `nextinstallmode` settings, and should therefore be used with due care.

5.4.5 Running Initialize Scripts

An *initialize script* is used when custom commands need to be executed before checking partitions and mounting devices (section 3.20.4). For example, to initialize some not explicitly supported hardware, or to do a RAID configuration lookup for a particular node. In such cases the custom commands are added to an `initialize` script. How to edit an `initialize` script is described in Appendix E.2.

An `initialize` script can be added to both a node's category and the node configuration. The node-installer first runs an `initialize` script, if it exists, from the node's category, and then an `initialize` script, if it exists, from the node's configuration.

The node-installer sets several environment variables which can be used by the `initialize` script. Appendix E contains an example script documenting these variables.

Related to the `initialize` script is the `finalize` script (section 5.4.11). This may run after node provisioning is done, but just before the `init` process on the node runs.

5.4.6 Checking Partitions, RAID Configuration, Mounting Filesystems

Behavior As Decided By The Install-Mode Value

In section 5.4.4 the node-installer determines the install-mode value, along with when to apply it to a node.

AUTO: The install-mode value is typically set to default to `AUTO`. If `AUTO` applies to the current node, it means the node-installer then checks the partitions of the local drive and its filesystems and recreates them in case of errors. Partitions are checked by comparing the partition layout of the local drive(s) against the drive layout as configured in the node's category configuration and the node configuration.

After the node-installer checks the drive(s) and, if required, recreates the layout, it mounts all filesystems to allow the drive contents to be synchronized with the contents of the software image.

FULL, MAIN, or SKIP: If install-mode values of `FULL`, `MAIN`, or `SKIP` apply to the current node instead, then no partition checking or filesystem checking is done by the node-installer.

NOSYNC: If the install-mode value of `NOSYNC` applies, then if the partition and filesystem checks both show no errors, the node starts up without getting an image synced to it from the provisioning node. If the partition or the filesystem check show errors, then the node partition is rewritten, and a known good image is synced across.

Behavior As Decided By XML Configuration Settings

The node-installer is capable of creating advanced drive layouts, including LVM setups, and hardware and software RAID setups. Drive layout examples and relevant documentation are in Appendix D.

The XML description used to set the drive layouts can be deployed for a single device or to a category of devices.

Hardware RAID: BCM supports hardware RAID levels 0, 1, 5, 10, and 50, and supports the following options:

- **stripe size:**

Option
64kB
128kB
256kB
512kB
1024kB

- **cache policy:**

Option
Cached
Direct

- **read policy:**

Option	Description
NORA	No Read Ahead
RA	Read Ahead
ADRA	Adaptive Read

- **write policy:**

Option	Description
WT	Write Through
WB	Write Back

5.4.7 Synchronizing The Local Drive With The Software Image

After having mounted the local filesystems, these can be synchronized with the contents of the software image associated with the node (through its category). Synchronization is skipped if the install-mode values of `NOSYNC` or `SKIP` are set, and takes place `FULL` or `AUTO` are set. Synchronization is delegated by the node-installer to the CMDaemon provisioning system. The node-installer just sends a provisioning request to CMDaemon on the head node.

For an install-mode of `FULL`, or for an install-mode of `AUTO` where the local filesystem is detected as being corrupted, full provisioning is done. For an install-mode of `AUTO` where the local filesystem is healthy and agrees with that of the software image, sync provisioning is done.

The `lock`, `unlock`, And `islocked` Commands For Software Images

The software image that is requested is available to nodes by default. Its availability can be altered and checked with the following commands:

- `lock`: this *locks* an image so that the image cannot be provisioned until the image is *unlocked*.

- `unlock`: this *unlocks* a locked image, so that request for provisioning the image is no longer prevented by a lock
- `islocked`: this lists the locked or unlocked states of images.

Locking an image is sometimes useful, for example, to make changes to an image when nodes are booting:

Example

```
[root@basecm11 ~]# cmsh
[basecm11]% softwareimage list
Name (key)           Path                                    Kernel version               Nodes
-------------------- --------------------------------------- ---------------------------- --------
default-image        /cm/images/default-image                3.10.0-1062.12.1.el7.x86_64  3
[basecm11]% softwareimage
[basecm11->softwareimage]% lock default-image
[basecm11->softwareimage]% islocked
Name           Locked
-------------- --------
default-image  yes
[basecm11->softwareimage]% lock default-image
[basecm11->softwareimage]% device
[basecm11->device]% reboot node001
    ...the cluster administrator makes changes to the node image during a boot, as it waits for
the image to unlock...
[basecm11->device]% softwareimage unlock default-image
```

For an unlocked image, on receiving the provisioning request, CMDaemon assigns the provisioning task to one of the provisioning nodes. The node-installer is notified when image synchronization starts, and also when the image synchronization task ends—whether it is completed successfully or not.

Exclude Lists: `excludelistsyncinstall` And `excludelistfullinstall`

What files are synchronized is decided by an *exclude list*. An exclude list is a property of the node category, and is a list of directories and files that are excluded from consideration during synchronization. The excluded list that is used is decided by the type of synchronization chosen: `full` or `sync`:

- A `full` type of synchronization rewrites the partition table of the node, then copies the filesystem from a software image to the node, using a list to specify files and directories to exclude from consideration when copying over the filesystem. The list of exclusions used is specified by the `excludelistfullinstall` property.

  The intention of `full` synchronization is to allow a complete working filesystem to be copied over from a known good software image to the node. By default the `excludelistfullinstall` list contains `/proc/`, `/sys/`, and `lost+found/`, which have no content in BCM's default software image. The list can be modified to suit the requirements of a cluster, but it is recommended to have the list adhere to the principle of allowing a complete working node filesystem to be copied over from a known good software image.

- A `sync` type of synchronization uses the property `excludelistsyncinstall` to specify what files and directories to exclude from consideration when copying parts of the filesystem from a known good software image to the node. The `excludelistsyncinstall` property is in the form of a list of exclusions, or more accurately in the form of two sub-lists.

  The contents of the sub-lists specify the parts of the filesystem that should be retained or not copied over from the software image during `sync` synchronization when the node is booting. The intention behind this is to have the node boot up quickly, updating only the files from the image

to the node that need updating due to the reboot of the node, and otherwise keeping files that are already on the node hard disk unchanged. The contents of the sub-lists are thus items such as the node log files, or items such as the /proc and /sys pseudo-filesystems which are generated during node boot.

The administrator should be aware that nothing on a node hard drive can be regarded as persistent because a FULL sync takes place if any error is noticed during a partition or filesystem check.

Anything already on the node that matches the content of these sub-lists is not overwritten by image content during an excludelistsyncinstall sync. However, image content that is not on the node is copied over to the node only for items matching the first sub-list. The remaining files and directories on the node, that is, the ones that are not in the sub-lists, lose their original contents, and are copied over from the software image.

A cmsh one-liner to get an exclude list for a category is:

```
cmsh -c "category use default; get excludelistfullinstall"
```

Similarly, to set the list:

```
cmsh -c "category use default; set excludelistfullinstall; commit"
```

where a text-editor opens up to allow changes to be made to the list. In Base View the navigation path is:
Grouping > Categories > Default > Settings > Exclude list full install
Image synchronization is done using rsync, and the syntax of the items in the exclude lists conforms to the "INCLUDE/EXCLUDE PATTERN RULES" section of the rsync(1) man page, which includes patterns such as "**", "?", and "[[:alpha:]]".

The excludelistfullinstall and excludelistsyncinstall properties decide how a node synchronizes to an image during boot. For a node that is already fully up, the related excludelistupdate property decides how a running node synchronizes to an image without a reboot event, and is discussed in section 5.6.

Interface Used To Receive Image Data: provisioninginterface
For regular nodes with multiple interfaces, one interface may be faster than the others. If so, it can be convenient to receive the image data using the fastest interface. Setting the value of provisioninginterface, which is a property of the node configuration, allows this.

By default it is set to BOOTIF for regular nodes. Using BOOTIF is not recommended for node configurations with multiple interfaces.

When listing the network interfaces in cmsh, the provisioning interface has a [prov] flag appended to its name.

Example

```
[basecm11->device[node001]->interfaces]% list
Type         Network device name  IP              Network
------------ -------------------- --------------- ----------------
physical     BOOTIF [prov]        10.141.0.1      internalnet
physical     eth1                 10.141.1.1      internalnet
physical     eth2                 10.141.2.1      internalnet
```

Head nodes and provisioninginterface**:** A head node in a single-head cluster does not use the provisioninginterface setting.

Head nodes in a failover configuration (Chapter 15), however, do have a value set for provisioninginterface, corresponding to the interface on the head that is being provisioned over internalnet by the other head (eth0 in figure 15.1).

Transport Protocol Used For Image Data: `provisioningtransport`
The `provisioningtransport` property of the node sets whether the image data is sent encrypted or unencrypted to the node from the provisioner. The property value is set using the `device` mode for the receiving node to one of these values:

- `rsyncdaemon`, which sends the data unencrypted
- `rsyncssh`, which sends the data encrypted

The `provisioningtransport` value can be set for all nodes, including provisioning nodes, head nodes, and cloud-director (section 3.2 of the *Cloudbursting Manual*) nodes. Because encryption severely increases the load on the provisioning node, using `rsyncssh` is only suggested if the users on the network cannot be trusted. By default, `provisioningtransport` is set to `rsyncdaemon`. If high availability (Chapter 15) is set up with the head nodes exposed to the outside world on the external network, the administrator should consider setting up `rsyncssh` for the head nodes.

The `rsyncssh` transport requires passwordless root access using `ssh` from the provisioner to the node being provisioned. This is configured by default in the default BCM nodes. However, if a new image is created with the `--exclude` options for `cm-create-image` as explained in (section 9.6.2), the keys must be copied over from `/root/.ssh/` on the existing nodes.

Tracking The Status Of Image Data Provisioning: `provisioningstatus`
The `provisioningstatus` command within the `softwareimage` mode of `cmsh` displays an updated state of the provisioning system. As a one-liner, it can be run as:

```
basecm11:~ # cmsh -c "softwareimage provisioningstatus"
Provisioning subsystem status
Pending request:            none
Provisioning node status:
+ basecm11
  Slots:                    0 / 10
  State:                    idle
  Active nodes:             none
  Up to date images:        default-image
```

The `provisioningstatus` command has several options that allow the requests to be tracked, including:

- `-r|--requestinfo`: displays the source and destination paths of provisioning requests,
- `-a|--all`: displays all status information on provisioning requests.

The Base View equivalent to `provisioningstatus` is accessed using the navigation path:
`Provisioning` > `Provisioning Node Status`

The Base View equivalent to `provisioningstatus -r` is accessed using the navigation path:
`Provisioning` > `Provisioning Request Status`

A provisioning request can be cancelled with the `cancelprovisioningrequest` command (section 5.2.4, page 255).

Tracking The Provisioning Log Changes: `synclog`
For a closer look into the image file changes carried out during provisioning requests, the `synclog` command from `device` mode can be used (lines elided in the following output):

Example

```
[basecm11->device]% synclog node001
Tue, 11 Jan 2011 13:27:17 CET - Starting rsync daemon based provisioning. Mode is SYNC.

sending incremental file list
./
...
deleting var/lib/ntp/etc/localtime
var/lib/ntp/var/run/ntp/
...
sent 2258383 bytes  received 6989 bytes  156232.55 bytes/sec
total size is 1797091769  speedup is 793.29

Tue, 11 Jan 2011 13:27:31 CET - Rsync completed.
```

Path Of The Provisioning Log File

The path of the log file can be found with the `-p` option:

Example

```
[basecm11->device]% synclog -p node001
/var/spool/cmd/rsync/node001-\.log
[basecm11->device]%
```

Statistical Analysis Of Provisioning Sessions: `syncinfo`

A provisioning session takes place between a provisioning image and a filesystem partition on a node. Statistics can be presented for the sessions using the `syncinfo` command. The statistical information presented is for number of files considered for transfer, the number of files that were actually transfered, how long the transfer took, which image and node were involved, and so on. The `syncinfo` command is run in device mode (output ellipsized and truncated):

Example

```
[head->device]% syncinfo
Node      Path                       Provisioner  Age   Duration Total files Transfered files ...
-------   ------------------------   -----------  ----  -------- ----------- ---------------- ...
node001   /cm/images/default-image   head         34s   21s      171,504     328              ...
node002   /cm/images/default-image   head         34s   22s      171,504     328              ...
...
```

The `syncinfo` command has options to run it per node, category, rack, and so on. Details on the options can be seen by running the `help` command (`help syncinfo`).

5.4.8 Writing Network Configuration Files

In the previous section, the local drive of the node is synchronized according to install-mode settings with the software image from the provisioning node. The node-installer now sets up configuration files for each configured network interface. These are files like:

`/etc/sysconfig/network-scripts/ifcfg-eth0`

for Red Hat, and Rocky Linux, while SUSE would use:

`/etc/sysconfig/network/ifcfg-eth0`

These files are placed on the local drive.

When the node-installer finishes its remaining tasks (sections 5.4.9–5.4.13) it brings down all network devices and hands over control to the local `/sbin/init` process. Eventually a local `init` script uses the network configuration files to bring the interfaces back up.

5.4.9 Creating A Local /etc/fstab File

The /etc/fstab file on the local drive contains local partitions on which filesystems are mounted as the init process runs. The actual drive layout is configured in the category configuration or the node configuration, so the node-installer is able to generate and place a valid local /etc/fstab file. In addition to all the mount points defined in the drive layout, several extra mount points can be added. These extra mount points, such as NFS imports, /proc, /sys and /dev/shm, can be defined and managed in the node's category and in the specific configuration of the node configuration, using Base View or cmsh (section 3.14.2).

5.4.10 Booting From The Local Hard Drive

By default, a node-installer boots from the software image on the head node using the network.

The node-installer can, optionally, during image synchronization, install a local drive boot record on the local hard drive if the installbootrecord boolean property of the node configuration or node category is set to on. Setting the local drive boot record means that the node tries to use a local hard drive boot installer during the next boot. This is a step toward having it become a standalone node that does not boot from the network. This step, and the other steps needed to allow booting from the local hard drive are covered next.

Setting The Boot Record To Allow The Node To Be Standalone

The local drive boot record is installed in the MBR of the local drive, overwriting the default iPXE boot record (section 5.1.2).

With a working custom software image, the boot record can be installed with cmsh commands for a node node001 with:

```
cmsh -c "device use node001; set installbootrecord yes; commit"
```

or for a category default with:

```
cmsh -c "category use default; set installbootrecord yes; commit"
```

Since installbootrecord is a boolean property, it means that if the node or if the node category have the value set, then the node uses that value.

In Base View, the equivalent is the Install boot record option. This can similarly be enabled and saved in the Base View node configuration or node category.

Setting the local drive boot record allows the next boot to be from the local hard drive, if the node is set up right to boot from the local hard drive.

Booting from the local hard drive often requires some further changes, as explained next.

Managing Boot Sequence And Bootloader To Ensure The Node Can Be Standalone

For a local hard drive boot to work:

1. hard drive booting must be set to have a higher priority than network booting in the BIOS of the node. Otherwise regular PXE booting is attempted, despite whatever value installbootrecord has.

2. A working bootloader must be present.

   By default, the node image for BCM has nodes set to use a SYSLINUX bootloader.

   If the administrator is not using the default software image, but is using a custom software image (section 9.6.1), and if the image is based on a running node filessystem that has not been built directly from a parent distribution, then the GRUB boot configuration may not be appropriate for a standalone GRUB boot to work. This is because the parent distribution installers often use special logic for setting up the GRUB boot configuration. Carrying out this same special logic for all distributions using the custom software image creation tool cm-create-image (section 9.6.2) is impractical.

Providing a custom working image from a standalone node that has been customized after direct installation from the parent distribution, ensures the GRUB boot configuration layout of the custom image is as expected by the parent distribution. This then allows a standalone GRUB boot on the node to run properly.

Nodes can be set to use a GRUB bootloader from within `device` mode, or from within `category` mode, by changing the `bootloader` parameter within the mode. For example, for a node `node001`:

Example

```
[root@basecm11 ~]# cmsh
[basecm11]% device use node001
[basecm11->device[node001]]% get bootloader
syslinux (default)
[basecm11->device[node001]]% set bootloader grub
[basecm11->device*[node001*]]% commit
```

or, for the `default` category:

```
[root@basecm11 ~]# cmsh
[basecm11]% category use default
[basecm11->category[default]]% get bootloader
syslinux
[basecm11->category[default]]% set bootloader grub
[basecm11->category*[default*]]% commit
```

Arranging for the two items in the preceding list ensures that the next boot is from GRUB on the hard drive. However, the `BOOTIF` also needs to be changed for booting to be successful. How it can be changed, and why it needs to be changed, is described next.

Changing `BOOTIF` To Ensure The Node Can Be Standalone

If the BIOS is set to boot from the hard drive, and if there is a working boot loader, and if the boot record has been installed, then the node boots using the boot record on the hard drive.

`BOOTIF` is the default value for the network interface for a node that is configured as a BCM software image. However, the `BOOTIF` interface is undefined during hard drive booting, because it depends on the network provisioning setup, which is not running. This means that the networking interface would fail during hard drive boot for a standard image. To remedy this, the interface should be set to a defined network device name, such as `eth0`, or the modern equivalents such as `en01` (section 5.8.1). The defined network device name, as the kernel sees it, can be found by logging into the node and taking a look at the output of `ip link`:

Example

```
[root@basecm11 ~]# ssh node001 ip link
1: lo: <LOOPBACK,UP,LOWER_UP> mtu 65536 qdisc noqueue...
2: eth0: <BROADCAST,MULTICAST,UP,LOWER_UP> mtu 1500 qdisc ...
```

In BCM the provisioning interface is mandatory, even if it is not provisioning. So it is set to the value of kernel-defined network device name instead of `BOOTIF`:

Example

```
[basecm11]% device interfaces node001
[basecm11->device[node001]->interfaces]% list
Type         Network device name  IP              Network
```

```
------------ ------------------- --------------- ----------------
physical     BOOTIF [prov]       10.141.0.1      internalnet
[basecm11->device[node001]->interfaces]% set bootif networkdevicename eth0
[basecm11->device*[node001*]->interfaces*]% commit
[basecm11->device[node001]->interfaces]% list
Type         Network device name  IP              Network
------------ ------------------- --------------- ----------------
physical     eth0 [prov]          10.141.0.1      internalnet
Tue Mar 31 13:46:44 2020 [notice] basecm11: node001 [   UP   ], restart required (eth0)
```

In the preceding example, the kernel-defined network device name is assumed to be `eth0`. It should be modified as required.

In addition, the new IP address is assumed to be on the same internal network. If the administrator intends the node to be standalone on another network, then the network and the IP address can be set to appropriate values.

When interface changes are carried out to make the node standalone, warnings show up saying that a reboot is required. A reboot of the node should be done when the interface configuration is complete.

During reboot, the node then boots from the hard drive as a standalone, with a non-`BOOTIF` network interface.

Bringing A Node That Boots From Its Hard Drive Back Into A Cluster

If the node is to be brought back into the cluster, then simply unsetting "Install boot record" and rebooting the node does not restore its iPXE boot record and hence its ability to iPXE boot. To restore the iPXE boot record, the node can be booted from the default image copy on the head node using a network boot again. Typically this is done by manual intervention during node boot to select network booting from the BIOS of the node.

Setting the value of `provisioninginterface` in `cmsh` for the node to `BOOTIF` is also recommended.

As suggested by the BCM iPXE boot prompt, setting network booting to work from the BIOS (regular "PXE" booting) is preferred to the relatively roundabout way of iPXE booting from the disk.

SELinux Initialization For Hard Drive Boot And PXE Boot

If configured, SELinux (Chapter 12 of the *Installation Manual*) is initialized at this point. For a boot from the hard drive, the initialization occurs if an SELinux filesystem has been saved to disk previously. For a PXE boot, the initialization takes place if the `SELinuxInitialize` directive is set to `true` in the `node-installer.conf` file.

5.4.11 Running Finalize Scripts

A *finalize script* is similar to an `initialize` script (section 5.4.5), only it runs a few stages later in the node-provisioning process.

In the context of configuration (section 3.20.4) it is used when custom commands need to be executed after the preceding mounting, provisioning, and housekeeping steps, but before handing over control to the node's local `init` process. For example, custom commands may be needed to:

- initialize some not explicitly supported hardware before `init` takes over
- supply a configuration file for the software image that cannot simply be added to the software image and used by `init` because it needs node-specific settings
- load a slightly altered standard software image on particular nodes, typically with the change depending on automatically detecting the hardware of the node it is being loaded onto. While this could also be done by creating a full new software image and loading it on to the nodes according to the hardware, it usually turns out to be better for simplicity's sake (future maintainability) to minimize the number of software images for the cluster.

The custom commands used to implement such changes are then added to the `finalize` script. How to edit a `finalize` script is described in Appendix E.2.

A `finalize` script can be added to both a node's category and the node configuration. The node-installer first runs a `finalize` script, if it exists, from the node's category, and then a `finalize` script, if it exists, from the node's configuration.

The node-installer sets several environment variables which can be used by the `finalize` script. Appendix E contains an example script which documents these variables.

5.4.12 Unloading Specific Drivers

Many kernel drivers are only required during the installation of the node. After installation they are not needed and can degrade node performance.

Baseboard Management Controllers (BMCs, section 3.7) that use IPMI drivers are an egregious example of this. The IPMI drivers are required to have the node-installer configure the IP address of any IPMI cards. Once the node is configured, these drivers are no longer needed, but they continue to consume significant CPU cycles and power if they stay loaded, which can affect job performance.

To solve this, the node-installer can be configured to unload a specified set of drivers just before it hands over control to the local init process. This is done by editing the `removeModulesBeforeInit` setting in the node-installer configuration file

`/cm/node-installer/scripts/node-installer.conf`,

For the `node-installer.conf` file in multidistro and multiarch (section 9.7) configurations, the directory path `/cm/node-installer` takes the form:

`/cm/node-installer-`<*distribution*>`-`<*architecture*>

The values for <*distribution*> and <*architecture*> can take the values outlined on page 558.

By default, the IPMI drivers are placed in the `removeModulesBeforeInit` setting.

To pick up IPMI-related data values, IPMI access is then carried out over the network without the drivers.

5.4.13 Switching To The Local `init` Process

At this point the node-installer is done. The node's local drive now contains a complete Linux installation and is ready to be started. The node-installer hands over control to the local `/sbin/init` process, which continues the boot process and starts all runlevel services. From here on the boot process continues as if the machine was started from the drive just like any other regular Linux machine.

5.5 Node States

During the boot process, several state change messages are sent to the head node CMDaemon or detected by polling from the head node CMDaemon. The most important node states for a cluster after boot up are introduced in section 2.1.1. These states are described again, along with some less common ones to give a more complete picture of node states.

5.5.1 Node States Icons In Base View

In the node icons used by Base View:

- Nodes in the `UP` state are indicated by an up-arrow.
  - If all health checks (section 10.2.4) for the node are successful, the up-arrow is green.
  - If there is a health check that fails or if the node requires a reboot, the up-arrow is red.
- Nodes in the `DOWN` state are indicated by a blue down-arrow.
- There are some other states, including:
  - Nodes in a `CLOSED` state are indicated by an X
  - Nodes in a `DOWN` state that are installing are indicated by a underscored down-arrow icon: ↓

5.5.2 Node States Shown In cmsh

In cmsh, the node state can be found using the status command from device mode for a node:

Example

```
[basecm11->device]% status -n node001..node002
node001 ..... [   UP   ] restart-required, health check failed
node002 ..... [  DOWN  ] (hostname changed) restart-required
```

The alias ds (page 49) is a convenient equivalent to device status and can be used anywhere within cmsh.

Devices in general can have their states conveniently listed with the list -f (page 61) command:

Example

```
[basecm11->device]% list -f "hostname:10, status:48"
hostname ( status
---------- ------------------------------------------------
apc01      [   UP   ]
basecm11   [   UP   ]
devhp      [   UP   ]
node001    [   UP   ] restart-required, health check failed
node002    [  DOWN  ] (hostname changed) restart-required
```

The reason for a red icon as shown in section 5.5.1 can be found within the parentheses. In this example it is (hostname changed).

5.5.3 Node States Indicating Regular Start Up

During a successful boot process the node goes through the following states:

- BOOTING. This is the state while the kernel and initrd are being downloaded by the node during network booting.

  To allow the BOOTING state to be detected for a node:

  - BOOTIF must be defined as an interface, or
  - if there is no interface with the value BOOTIF, but a particular network device, such as eth0, is the boot interface, then a special revision tag of bootif can be used for no more than one interface:

    Example

    ```
    [basecm11->device[node001]->interfaces[eth0]]% set revision bootif
    ```

- INSTALLING. This state is normally entered as soon as the node-installer has determined on which node the node-installer is running. Within this state, information messages display indicating what is being done while the node is in the INSTALLING state. Possible messages under the status column for the node within cmsh and Base View are normally, in sequence:

  1. node-installer started
  2. Optionally, the following two messages:
     (a) waiting for user input
     (b) installation was resumed
  3. checking disks
  4. recreating partitions and filesystems
  5. mounting disks

6. One of these following two messages:
   (a) `waiting for FULL provisioning to start`
   (b) `waiting for SYNC provisioning to start`
7. `provisioning started, waiting for completion`
8. `provisioning complete`
9. `initializing SELinux`

Between steps 1 and 3 in the preceding, these optional messages can also show up:

- If `burn` mode is entered or left:

```
running burn-in tests
burn-in test completed successfully
```

- If `maintenance` mode is entered:

```
entered maintenance mode
```

- `INSTALLER_CALLINGINIT`. This state is entered as soon as the node-installer has handed over control to the local `init` process. The associated message normally seen with it in `cmsh` and Base View is:
  - `switching to local root`
- `UP`. This state is entered as soon as the CMDaemon of the node connects to the head node CMDaemon.

5.5.4 Node States That May Indicate Problems

Other node states are often associated with problems in the boot process:

- `DOWN`. This state is registered as soon as the CMDaemon on the regular node is no longer detected by CMDaemon on the head node. In this state, the state of the regular node is still tracked, so that CMDaemon is aware if the node state changes.
- `CLOSED`. This state is appended to the `UP` or `DOWN` state of the regular node by the administrator, and causes most CMDaemon monitoring actions for the node to cease. The state of the node is however still tracked by default, so that CMDaemon is aware if the node state changes.
  - The `close` command can be used to set the `CLOSED` state from within `device` mode of `cmsh`. The `help` text for the command gives details on how it can be applied to categories, groups and so on. The `-m` option sets a message by the administrator for the closed node or nodes.

    Example

```
[root@headymcheadface ~]# cmsh
[headymcheadface]% device
[headymcheadface->device]% close -m "fan dead" -n node001,node009,node020
Mon May 2 16:32:01 [notice] headymcheadface: node001 ...[ DOWN/CLOSED ] (fan dead)
Mon May 2 16:32:01 [notice] headymcheadface: node009 ...[ DOWN/CLOSED ] (fan dead)
Mon May 2 16:32:01 [notice] headymcheadface: node020 ...[ DOWN/CLOSED ] (fan dead)
```

    The `CLOSED` state can also be set from Base View using the navigation path

    `Devices > All Devices > ⋮ > Monitoring > Monitored State > Close`

    When the `CLOSED` state is set for a device, CMDaemon commands can still attempt to act upon it. For example, in the `device` mode of `cmsh`, the `open` command, described next, can act on it.

- open: This is the converse to the close command. It has the same options, including the -m option that logs a message. It also has the following extra options:
  - --reset: Resets whatever the status is of the devicestatus check. However, this reset by itself does not solve any underlying issue. The issue may still require a fix, despite the status having been reset.
    For example, the --reset option can be used to reset the restart-required flag (section 5.5.2). However, the reason that set the restart-required flag is not solved by the reset. Restarts are required for regular nodes if there have been changes in the following: network settings, disk setup, software image, or category.
  - -f|--failbeforedown <*count*>: Specifies the number of failed pings before a device is marked as down (default is 1).
- drain and undrain (Appendix G.4.1)
- For nodes that have power control:
  - power -f on
  - power -f off
  - power -f reset

  Power control mechanisms such as PDUs, custom power scripts, and BMCs using IPMI/HP iLO/DRAC/CIMC/Redfish, are described in Chapter 4.

- The mute command leaves the device monitored and in its regular UP or DOWN state, but suppresses the monitoring actions (such as alerts and triggers) that would otherwise be acting. This is useful, for example, while maintenance is carried out on a node, so that expected failures do not generate notifications. The -m option, in a way similar to the how it works for the close command, adds a message that is displayed in the device status, and node selection options such as -n, -c, and -r select the devices to act on.
- The unmute command is the converse of the mute, and re-enables monitoring actions.

Example

```
[basecm11->device]% mute -m "firmware upgrade in progress" -n node001
muted node001
[basecm11->device]% status node001
node001 ................. [   UP   ] (firmware upgrade in progress), muted
[basecm11->device]% unmute -n node001
unmuted node001
```

In Base View, the equivalents for a node node001 for example, are using the navigation paths:

- Devices > All Devices > node001 > Monitored state > Open|Close|Mute|Unmute
- Devices > All Devices > node001 > Workload > Drain Overview|Drain|Undrain
- Devices > All Devices > node001 > Power > On|Off|Reset.

CMDaemon on the head node only maintains device monitoring logs for a device that is in the UP state. If the device is in a state other than UP, then CMDaemon only tracks its state, and can display the state if queried.

For example: if a node displays the state UP when queried about its state, and is given a 'close' command, it then goes into a CLOSED state. Querying the node state then displays the state UP/CLOSED. It remains in that CLOSED state when the node is powered down. Querying the node state after

being powered down displays DOWN/CLOSED. Next, powering up the node from that state, and having it go through the boot process, has the node displaying various CLOSED states during queries. Typically the query responses show it transitioning from DOWN/CLOSED, to INSTALLING/CLOSED, to INSTALLER_CALLINGINIT/CLOSED, and ending up displaying the UP/CLOSED state.

Thus, a node set to a CLOSED state remains in a CLOSED state regardless of whether the node is in an UP or DOWN state. The only way out of a CLOSED state is for the administrator to tell the node to open via the cmsh "open" option discussed earlier. The node, as far as CMDaemon is concerned, then switches from the CLOSED state to the OPEN state. Whether the node listens or not does not matter—the head node records it as being in an OPENING state for a short time, and during this time the next OPEN state (UP/OPEN, DOWN/OPEN, etc.) is agreed upon by the head node and the node.

When querying the state of a node, an OPEN tag is not displayed in the response, because it is the "standard" state. For example, UP is displayed rather than UP/OPEN. In contrast, a CLOSED tag is displayed when it is active, because it is a "special" state.

The CLOSED state is normally set to take a node that is unhealthy out of the cluster management system. The node can then still be in the UP state, displaying UP/CLOSED. It can even continue running workload jobs in this state, since workload managers run independent of CMDaemon. So, if the workload manager is still running, the jobs themselves are still handled by the workload manager, even if CMDaemon is no longer aware of the node state until the node is re-opened. For this reason, draining a node is often done before closing a node, although it is not obligatory.

- OPENING. This transitional state is entered as soon as the CMDaemon of the node rescinds the CLOSED state with an "open" command from cmsh. The state usually lasts no more than about 5 seconds, and never more than 30 seconds in the default configuration settings of BCM. The help text for the open command of cmsh gives details on its options.

- INSTALLER_FAILED. This state is entered from the INSTALLING state when the node-installer has detected an unrecoverable problem during the boot process. For instance, it cannot find the local drive, or a network interface cannot be started. This state can also be entered from the INSTALLER_CALLINGINIT state when the node takes too long to enter the UP state. This could indicate that handing over control to the local init process failed, or the local init process was not able to start the CMDaemon on the node. Lastly, this state can be entered when the previous state was INSTALLER_REBOOTING and the reboot takes too long.

- INSTALLER_UNREACHABLE. This state is entered from the INSTALLING state when the head node CMDaemon can no longer ping the node. It could indicate the node has crashed while running the node-installer.

- INSTALLER_REBOOTING. In some cases the node-installer has to reboot the node to load the correct kernel. Before rebooting it sets this state. If the subsequent reboot takes too long, the head node CMDaemon sets the state to INSTALLER_FAILED.

5.6 Updating Running Nodes

Updating Running Nodes From A Stored Image By Rebooting

Changes made to the contents of the software image for nodes, kept on the head node, become a part of any other provisioning nodes according to the housekeeping system on the head node (section 5.2.4).

Thus, when a regular node reboots, the latest image is installed from the provisioning system onto the regular node via a provisioning request (section 5.4.7).

Updating Running Nodes From A Stored Image Without Rebooting

However, updating a running node with the latest software image changes is also possible without rebooting it. Such an update can be requested using cmsh or Base View, and is queued and delegated to a provisioning node, just like a regular provisioning request. The properties that apply to

the regular provisioning of an image also apply to such an update. For example, the value of the `provisioninginterface` setting (section 5.4.7) on the node being updated determines which interface is used to receive the image.

- In `cmsh` the request is submitted with the `imageupdate` option (section 5.6.2).
- In Base View, it is submitted, for a node `node001` for example, using the navigation path:

  `Devices` > All Devices > node001 ⋮ > `Software Image` > `Update Node` (section 5.6.3).

The `imageupdate` command of `cmsh` and the `Update Node` menu option of Base View use a configuration file called `excludelistupdate`, which is, as its name suggests, a list of exclusions to the update.

The running node is thus updated from a stored image with the help of that configuration file when `imageupdate` or `Update Node` are run. More details are given in the rest of this section (section 5.6).

Updating A Stored Image From A Running Node

The converse, that is, to update a stored image from what is on a running node, can be also be carried out. This converse can be viewed as grabbing from a node, and synchronizing what is grabbed, to an image. It can be done using `grabimage` (`cmsh`), or `Grab to image` (Base View), and involves further exclude lists `excludelistgrab` or `excludelistgrabnew`. The `grabimage` command and `Grab to image` option are covered in detail in section 9.5.2.

5.6.1 Updating Running Nodes: Configuration With `excludelistupdate`

The exclude list `excludelistupdate` used by the `imageupdate` command is defined as a property of the node's category. It has the same structure and `rsync` patterns syntax as that used by the exclude lists for provisioning the nodes during installation (section 5.4.7).

Distinguishing Between The Intention Behind The Various Exclude Lists

The administrator should note that it is the `excludelist`*update* list that is being discussed here, in contrast with the `excludelistsync`*install*/`excludelistfull`*install* lists which are discussed in section 5.4.7, and also in contrast with the `excludelist`*grab*/`excludelist`*grabnew* lists of section 9.5.2.

So, for the `imageupdate` command the `excludelistupdate` list concerns an *update* to a running system, while for installation `sync` or `full` provisioning, the corresponding exclude lists (`excludelistsyncinstall` and `excludelistfullinstall`) from section 5.4.7 are about an *install* during node start-up. Because the copying intention during updates is to be speedy, the `imageupdate` command synchronizes files rather than unnecessarily overwriting unchanged files. Thus, the `excludelistupdate` exclusion list it uses is actually analogous to the `excludelistsyncinstall` exclusion list used in the `sync` case of section 5.4.7, rather than being analogous to the `excludelistfullinstall` list.

Similarly, the `excludelistgrab`/`excludelistgrabnew` lists of section 9.5.2 are about a *grab* from the running node to the image.

- The `excludelistgrab` list here is intended for the case of synchronizing the existing image with the running node, and is thus analogous to the `excludelistsyncinstall` exclusion list.
- The `excludelistgrabnew` list here is intended for the case of copying a full image from the running node, and is thus analogous to the `excludelistfullinstall` list.

The following table summarizes this:

During:	Exclude list used is:	Copy intention:
update	excludelistupdate	sync, image to running node
install	excludelistfullinstall	full, image to starting node
	excludelistsyncinstall	sync, image to starting node
grab	excludelistgrabnew	full, running node to image
	excludelistgrab	sync, running node to image

The preceding table is rather terse. It may help to understand it if is expanded with some in-place footnotes, where the footnotes indicate what actions can cause the use of the exclude lists:

During:	Exclude list used is:	Copy intention:
update eg: imageupdate	excludelistupdate	sync, image to running node
install eg: node-provisioning process during pre-init stage depending on installmode decision	excludelistfullinstall eg: node provisioning with installmode FULL	full, image to starting node
	excludelistsyncinstall eg: node provisioning AUTO with healthy partition	sync, image to starting node
grab eg: grabimage (cmsh), Grab to image (Base View)	excludelistgrabnew grabimage -i/Grab to image for a new image	full, running node to image
	excludelistgrab grabimage/Grab to image for the original image	sync, running node to image

The Exclude List Logic For excludelistupdate

During an imageupdate command, the synchronization process uses the excludelistupdate list, which is a list of files and directories. One of the cross checking actions that may run during the synchronization is that the items on the list are excluded when copying parts of the filesystem from a known good software image to the node. The detailed behavior is as follows:

The exludelistupdate list is in the form of two sublists. Both sublists are lists of paths, except that the second sublist is prefixed with the text "no-new-files: " (without the double quotes). For the node being updated, all of its files are looked at during an imageupdate synchronization run. During such a run, the logic that is followed is:

- if an excluded path from excludelistupdate exists on the node, then nothing from that path is copied over from the software image to the node
- if an excluded path from excludelistupdate does not exist on the node, then
  - if the path is on the first, non-prefixed list, then the path is copied over from the software image to the node.
  - if the path is on the second, prefixed list, then the path is not copied over from the software image to the node. That is, no new files are copied over, like the prefix text implies.

This is illustrated by figure 5.18.

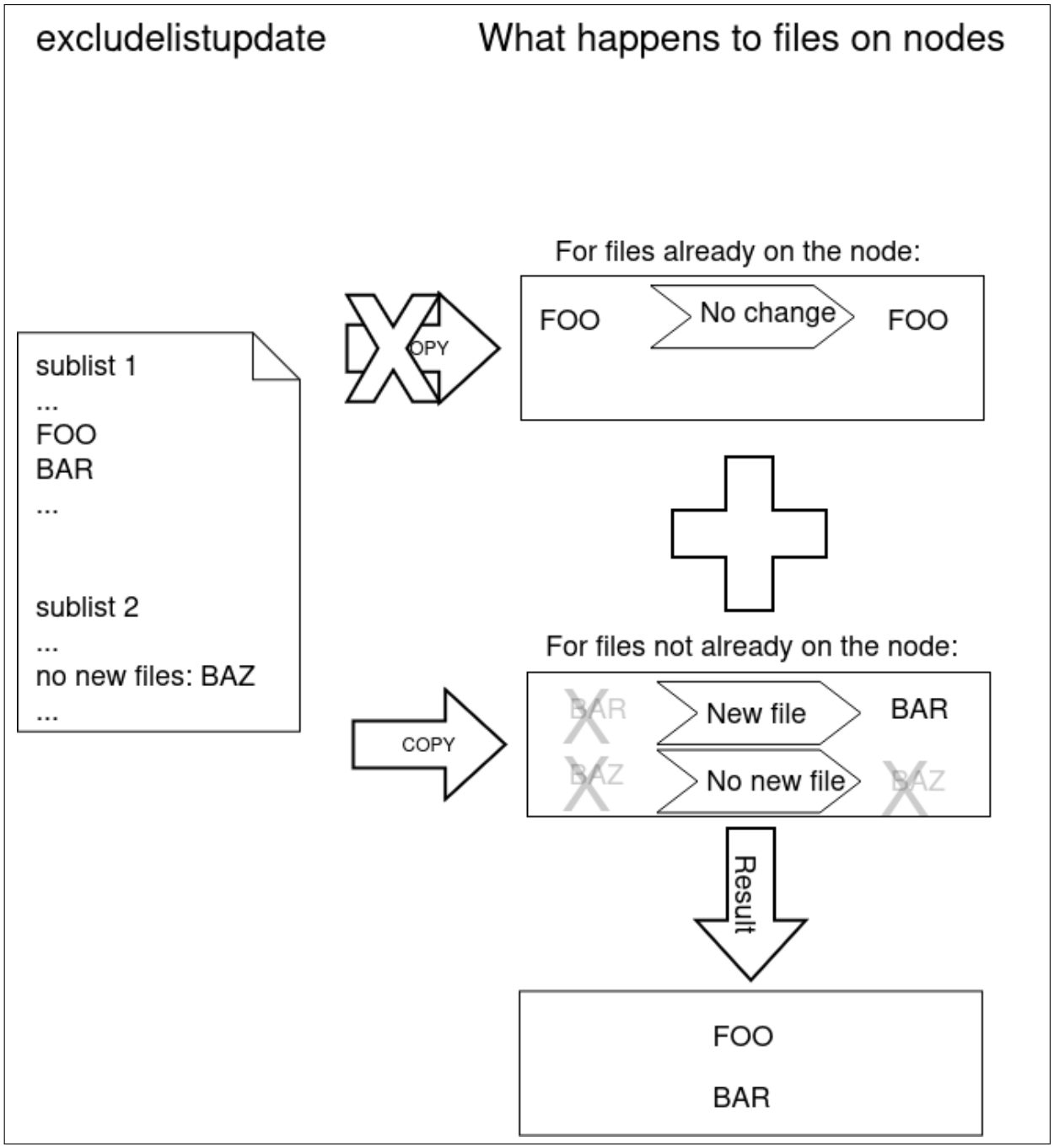


Figure 5.18: Exclude list logic

The files and directories on the node that are not in the sub-lists lose their original contents, and are copied over from the software image. So, content not covered by the sub-lists at that time is normally not protected from deletion.

Thus, the provisioning system excludes paths described according to the `excludelistupdate` property.

The provisioning system also excludes a statically-imported filesystem on a node if the filesystem is a member of the following special list: NFS, Lustre, FUSE, CephFS, CIFS, PanFS, FhGFS, BeeGFS, GlusterFS, or GPFS. If this exclusion were not done, then all data on these imported filesystems would be wiped, since they are not part of the software image. The automatic exclusion for these imported

filesystems does not rely on the excludelist values maintained by CMDaemon—instead, CMDaemon carries out the check on-the-fly when provisioning starts.

Statically-imported filesystems that have their mounts managed by BCM via the `fsmounts` mode can be excluded from being mounted on the nodes in the first place, by removing them from the listed mounts within the `fsmounts` mode.

Imported filesystems not on the special list can have their data wiped out during provisioning or sync updates, if the statically-imported filesystems are placed in the image manually—that is, if the filesystems are mounted manually into the image on the head node via `/etc/fstab` without using `cmsh` or Base View.

Filesystems mounted dynamically cannot have their appearance or disappearance detected reliably: Any filesystem that may be imported via an auto-mount operation must therefore explicitly be excluded by the administrator manually adding the filesystem to the exclude list. This is to prevent an incorrect execution of `imageupdate`. Neglecting to do this may wipe out the filesystem, if it happens to be mounted in the middle of an `imageupdate` operation.

The `fstab` system is a statically mounting system, and not an auto-mounter: While `fstab` mounts filesystems automatically, system administrators should not confuse that with auto-mounting. Automounting as provided by `autofs` is designed for the dynamic mounting of filesystems on demand by regular users. The `fstab` table is designed for mounting as carried out by the system, as occurs during boot, which is why it is regarded as a static, non-auto-mounting system.

Editing An Exclude List

A sample `cmsh` one-liner which opens up a text editor in a category so that the exclude list for updates can be edited is:

```
cmsh -c "category use default; set excludelistupdate; commit"
```

Similarly, the exclude list for updates can also be edited in Base View using the navigation path:

`Grouping > Categories`  `> Settings > Exclude list update`

Provisioning Modifications Via `excludelistmanipulatescript`

Sometimes the administrator has a need to slightly modify the execution of exclude lists during provisioning. The `excludelistmanipulatescript` file takes as an input the exclude list inherited from a category, modifies it in some way, and then produces a new exclude list. Conceptually it is a bit like how an administrator might use `sed` if it worked without a pipe. As usual, setting it for node level overrides the category level.

A script that manipulates the exclude lists of a node can be specified as follows within `cmsh`:

```
[basecm11]% device use node001
[basecm11->device[node001]]% set excludelistmanipulatescript
(a vi session will start. A script is edited and saved)
[basecm11->device[node001*]]% commit
```

The script can be as simple as:

Example

```
#!/bin/bash

echo "- *"
echo 'no-new-files: - *'
```

If provisioning a node from the head node, then the script modifies the node-provisioning exclude lists—excludelistfullinstall, excludelistsyncinstall, and excludelistupdate—so that they appear to contain these items only:

```
- *
no-new-files: - *
```

The provisioning node then excludes everything during provisioning.

Careful editing and testing of the script is advised. Saving a script with just a single whitespace, for example, usually has undesirable results.

A more useful script template is the following:

Example

```
#!/bin/bash

while read; do
  echo "$REPLY"
done

echo "# This and next line added by category excludelistmanipulatescript."
echo "# The command line arguments were: $@"
```

The provisioning exclude lists are simply read in, then sent out again without any change, except that the last two lines add comment lines to the exclude lists that are to be used.

Internally, the arguments taken by the excludelistmanipulatescript are the destination path and the sync mode (one of install|update|full|grab|grabnew). This can be seen in the output of $@, if running an imageupdate command to execute a dry run with the preceding example:

```
[basecm11]% device use node001
[basecm11->device[node001]]% get excludelistmanipulatescript
 (the script is put in)
[basecm11->device[node001*]]% commit; imageupdate
Performing dry run (use synclog command to review result, then pass -w to perform real update)...
Wed Apr 15 04:55:46 2015 [notice] basecm11: Provisioning started: sendi\
ng basecm11:/cm/images/default-image to node001:/, mode UPDATE, dry run\
 = yes, no data changes!
[basecm11->device[node001]]%
Wed Apr 15 04:55:51 2015 [notice] basecm11: Provisioning completed: sen\
t basecm11:/cm/images/default-image to node001:/, mode UPDATE, dry run \
= yes, no data changes!
imageupdate [ COMPLETED ]
```

An excerpt from the sync log, after running the synclog command, then shows output similar to (some output elided):

```
...
- /cm/shared/*
- /cm/shared/
- /home/*
- /home/
- /cm/shared/apps/slurm/*
- /cm/shared/apps/slurm/
# This and next line added by category excludelistmanipulatescript.
# The command line arguments were: update /
```

```
Rsync output:
sending incremental file list
cm/local/apps/cmd/scripts/healthchecks/configfiles/
...
```

Here, the sync mode is `update` and the destination path is "/". Which of the exclude lists is being modified can be determined by the `excludelistmanipulatescript` by parsing the sync mode.

The bash variable that accepts the exclude list text is set to a safely-marked form using curly braces. This is done to avoid expansion surprises, due to wild card characters in the exclude lists. For example, if `$REPLY` were used instead of `${REPLY}`, and the script were to accept an exclude list line containing "`-/proc/*`", then it would give quite confusing output.

Other Exclude List Handling Options

The `excludelistfailover` and `excludelistnormal` files: are two further exclude list files that modify standard provisioning behavior. These are discussed in section 15.4.8.

The `excludelistsnippets` tool: When synchronizing to a cloud director, or to an edge director, it is sometimes useful to exclude unneeded files and paths from the synchronization, in order to speed it up. The `excludelistsmanipulatescript` tool is powerful enough to do it, but it has some issues due to its power. For example, it is a script, which means that it is called whenever it is used, and so uses up some extra resources. Also, it is a bit tricky to set up.

An easier way to manipulate exclude lists for the unneeded files and paths is via the `excludelistsnippets` tool, described in section 4.3.1 of the *Cloudbursting Manual*. This tool allows additional exclusion to be specified in a simpler way.

Exclude List State At Node Level

Exclude lists at category level and node level: An exclude list can be set at node level, as well as at category level. Roles and overlays can add implied exclude lists too.

At category level, an exclude list such as `excludelistfullinstall` can be set up explicitly with:

Example

```
[basecm11->category[default]]% set excludelistfullinstall
...a text editor such as vi opens up and the list can be edited...
[basecm11->category*[default*]]% commit
```

At node level, an exclude list can be set in the same way:

Example

```
[basecm11->device[node001]]% set excludelistfullinstall
...a text editor such as vi opens up and the list can be edited...
[basecm11->device*[node001*]]% commit
```

An exclude list that is not empty at node level overrules its corresponding category list. Exclude lists brought in via roles are however simply included in the exclude list.

The `excludelist` command: At node level it can be unclear what the resulting exclude list ("operational exclude list") actually is. The exclude list state at node level can therefore be viewed using the `excludelist` command options. The `excludelist` command becomes active if a software image has been set at the node level.

- The `list` option to `excludelist` lists the source and destination paths:

Example

```
[basecm11->device[node001]]% excludelist list
Source path (on the head node)   Destination path (on the node)
-------------------------------- --------------------------------
/cm/images/default-image         /
```

- The `get` option to the `excludelist` command has synchronization mode and destination suboptions for a node.

  Earlier on (page 295), the intention behind the various exclude lists, according to the type of update or synchronization, were distinguished.

  The `excludelist get` command can have a destination path specified, and have the type of update or synchronization specified according to those distinguishing concepts.

  The output to the `excludelist get` command then shows the operational exclude list as seen by a node for that path and for that update or synchronization.

  Thus, for example:

  – The `full` install operational exclude list for the path `/` on node `node001`, intended for a full installation to a node that is starting up, can be found as follows:

  Example

  ```
  [basecm11->device[node001]]% excludelist get full /
  # For details on the exclude patterns defined here please refer to
  # the FILTER RULES section of the rsync man page.
  #
  # Files that match these patterns will not be installed onto the node.
  - lost+found/
  - /proc/*
  - /sys/*
  - /boot/efi


  # extra defaults
  - /proc/*
  - /sys/*
  ```

  – Similarly, the `sync` install operational exclude list for the path `/` on node `node001`, intended for a sync installation to a node that is starting up, can be found as follows:

  ```
  [basecm11->device[node001]]% excludelist get sync /

  # For details on the exclude patterns defined here please refer to
  # the FILTER RULES section of the rsync man page.
  #
  # Files that exist on a node and match one of these patterns will not be
  # modified or deleted. Any files that match one of these patterns and that
  # exist in the image but are absent on the node, will be copied to the node.
  - /.autofsck
  - /boot/grub*/grub.cfg
  - /cm/local/apps/openldap/etc/certs/ldap.key
  - /cm/local/apps/openldap/etc/certs/ldap.pem
  - /data/*
  - /home/*
  ```

...

– Other `excludelist get` options, besides `full` and `sync`, are:
  * `grab` (a grab from a running node for a sync back to an existing image)
  * `grabnew` (a grab from a running node for a full install to a new image)
  * `update` (a sync update of a running node from an image).

  All `excludelist get` options correspond to the intentions of the associated exclude list types as distinguished on page 295.

5.6.2 Updating Running Nodes: With `cmsh` Using `imageupdate`

Using a defined `excludelistupdate` property (section 5.6.1), the `imageupdate` command of `cmsh` is used to start an update on a running node:

Example

```
[basecm11->device]% imageupdate -n node001
Performing dry run (use synclog command to review result, then pass -w to perform real update)...
Tue Jan 11 12:13:33 2011 basecm11: Provisioning started on node node001
[basecm11->device]% imageupdate -n node001: image update in progress ...
[basecm11->device]%
Tue Jan 11 12:13:44 2011 basecm11: Provisioning completed on node node001
```

By default the `imageupdate` command performs a dry run, which means no data on the node is actually written. Before passing the "`-w`" switch, it is recommended to analyze the rsync output using the `synclog` command (section 5.4.7).

If the user is now satisfied with the changes that are to be made, the `imageupdate` command is invoked again with the "`-w`" switch to implement them:

Example

```
[basecm11->device]% imageupdate -n node001 -w
Provisioning started on node node001
node001: image update in progress ...
[basecm11->device]% Provisioning completed on node node001
```

5.6.3 Updating Running Nodes: With Base View Using the `Update node` Option

In Base View, an image update can be carried out by selecting the specific node or category, for example `node001`, and updating it via the navigation path:

`Devices > All Devices > node001` ⋮ `> Software Image > Update Node`

5.6.4 Updating Running Nodes: Considerations

An attempt to update the image on a running node can run into some issues:

- Updating an image via `cmsh` or Base View automatically updates the provisioners first via the `updateprovisioners` command (section 5.2.4) if the provisioners have not been updated in the last 5 minutes. The conditional update period can be set with the `dirtyautoupdatetimeout` parameter (section 5.2.4).

  So, with the default setting of 5 minutes, if there has been a new image created within the last 5 minutes, then provisioners do not get the updated image when doing the updates, which means that nodes in turn do not get those updates. Running the `updateprovisioners` command just before running the `imageupdate` command therefore usually makes sense.

- By default, BCM does not allow provisioning if automount (page 932) is running.
- Also, when updating services, the services on the nodes may not restart since the `init` process may not notice the replacement.

For these reasons, especially for more extensive changes, it can be safer for the administrator to simply reboot the nodes instead of using `imageupdate` to provision the images to the nodes. A reboot by default ensures that a node places the latest image with an `AUTO` install (section 5.4.7), and restarts all services.

The `Reinstall node` option, which can be run, for example, on a node `node001`, using a navigation path of:

`Devices > All Devices > node001` ⋮ `> Software image > Reinstall node`

also does the same as a reboot with default settings, except for that it unconditionally places the latest image with a `FULL` install, and so may take longer to complete.

5.7 Adding New Nodes

How the administrator can add a single node to a cluster is described in section 1.3 of the *Installation Manual*. This section explains how nodes can be added in ways that are more convenient for larger numbers of nodes.

5.7.1 Adding New Nodes With `cmsh` And Base View Add Functions

Node objects can be added from within the `device` mode of `cmsh` by running the `add` command:

Example

```
[basecm11->device]% add physicalnode node002 10.141.0.2
[basecm11->device*[node002*]% commit
```

The Base View equivalent of this can be carried out using the navigation path:
`Devices > All Devices > + Add > PhysicalNode > Hostname`
then adding the value `node002` to `Hostname`, and saving it.

When adding the node objects in `cmsh` and Base View, some values (the MAC addresses for example) may need to be filled in before the object validates. For regular nodes, there should be an interface and an IP address for the network that it boots from, as well as for the network that manages the nodes. A regular node typically has only one interface, which means that the same interface provides boot and management services. This interface is then the boot interface, BOOTIF, during the pre-init stage, but is also the management interface, typically `eth0` or whatever the device is called, after the pre-init stage. The IP address for BOOTIF is normally provided using DHCP, while the IP address for the management interface can be set to a static IP address using `cmsh` or Base View by the administrator.

Adding new node objects as "placeholders" can also be done from `cmsh` or Base View. By placeholders, here it is meant that an incomplete node object is set. For example, sometimes it is useful to create a node object with the MAC address setting unfilled because it is still unknown. Why this can be useful is covered shortly.

5.7.2 Adding New Nodes With The Node Creation Wizard

Besides adding nodes using the `add` command of `cmsh` or the `+ Add` option of Base View as in the preceding text, there is also a Base View wizard that guides the administrator through the process—the *node creation wizard*. This is useful when adding many nodes at a time. It is available using the navigation path:

`Devices > Node Creation Wizard`

This device creation wizard should not be confused with the somewhat-related device *identification* resource described in section 5.4.2, which identifies unassigned MAC addresses and switch ports, and helps assign them node names.

- The node *creation* wizard creates an object for nodes, assigns them node names, but it leaves the MAC address field for these nodes unfilled, keeping the node object as a "placeholder".
- The device *identification* resource assigns MAC addresses so that node names are associated with a MAC address.

If a node is left with an unassigned MAC address—that is, in a "placeholder" state—then it means that when the node starts up, the provisioning system lets the administrator associate a MAC address and switch port number at the node console for the node. This occurs when the node-installer reaches the node configuration stage during node boot as described in section 5.4.2. This is sometimes preferable to associating the node name with a MAC address remotely with the device identification resource.

The node creation wizard goes through the following screens:

- The `Node Range and Category` screen: An example of this screen is seen in figure 5.19, where a cluster with existing nodes `node001`, `node002`, and `node003` has 10 more node objects added to it, with numbering set to start from `node004` onwards.
  - The `Node Range` section: allows setting of the number of nodes to be added, their node name prefix, and their numeric suffix.
  - The `Category, Management Network and Cloud Provider` section: allows setting of the node category for the nodes, their management network, and their cloud provider, if any.

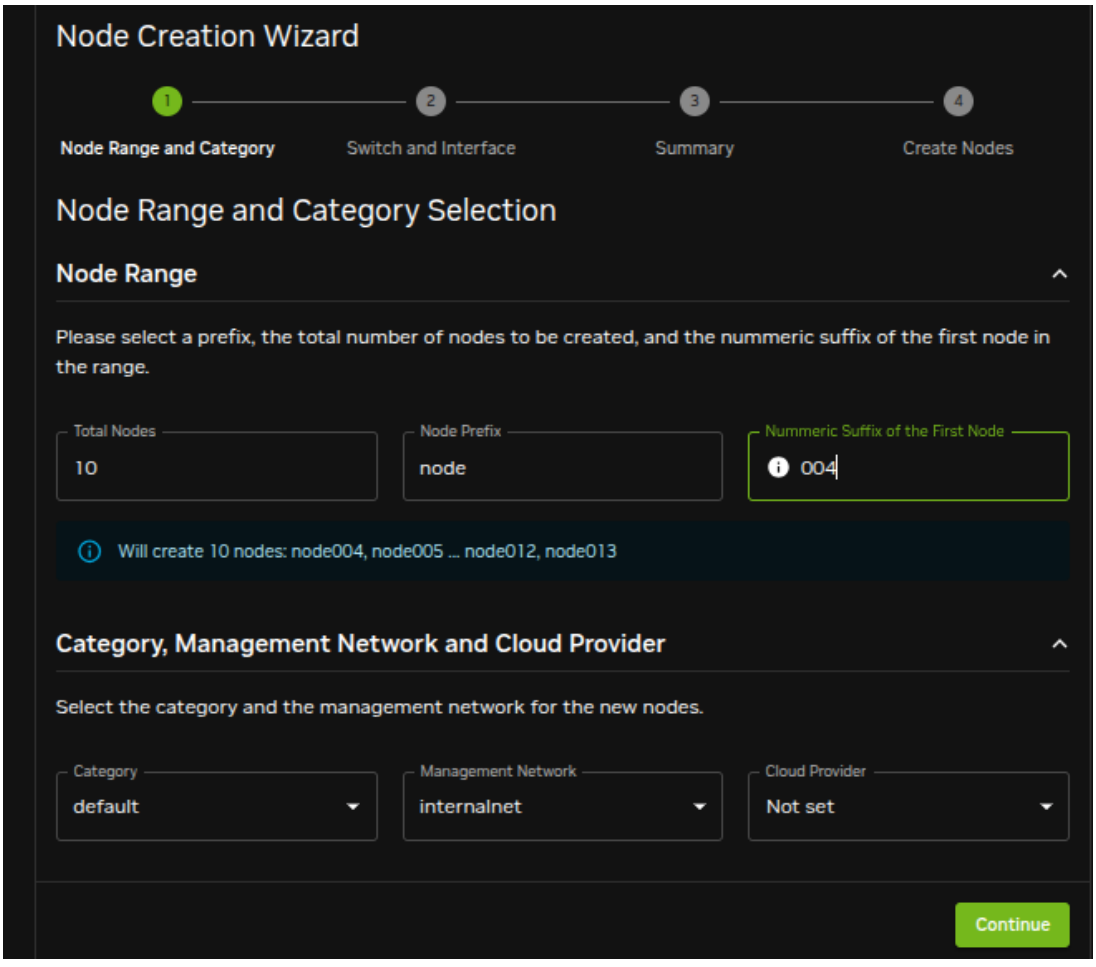


Figure 5.19: Node Creation Wizard: Node Range And Category Screen

Clicking on the `Continue` button brings up the next screen.

- The `Switch and Interface` screen: lists the nodes and allows their interface properties to be assigned in bulk with the `Assign Interfaces in Bulk` option (figure 5.20):

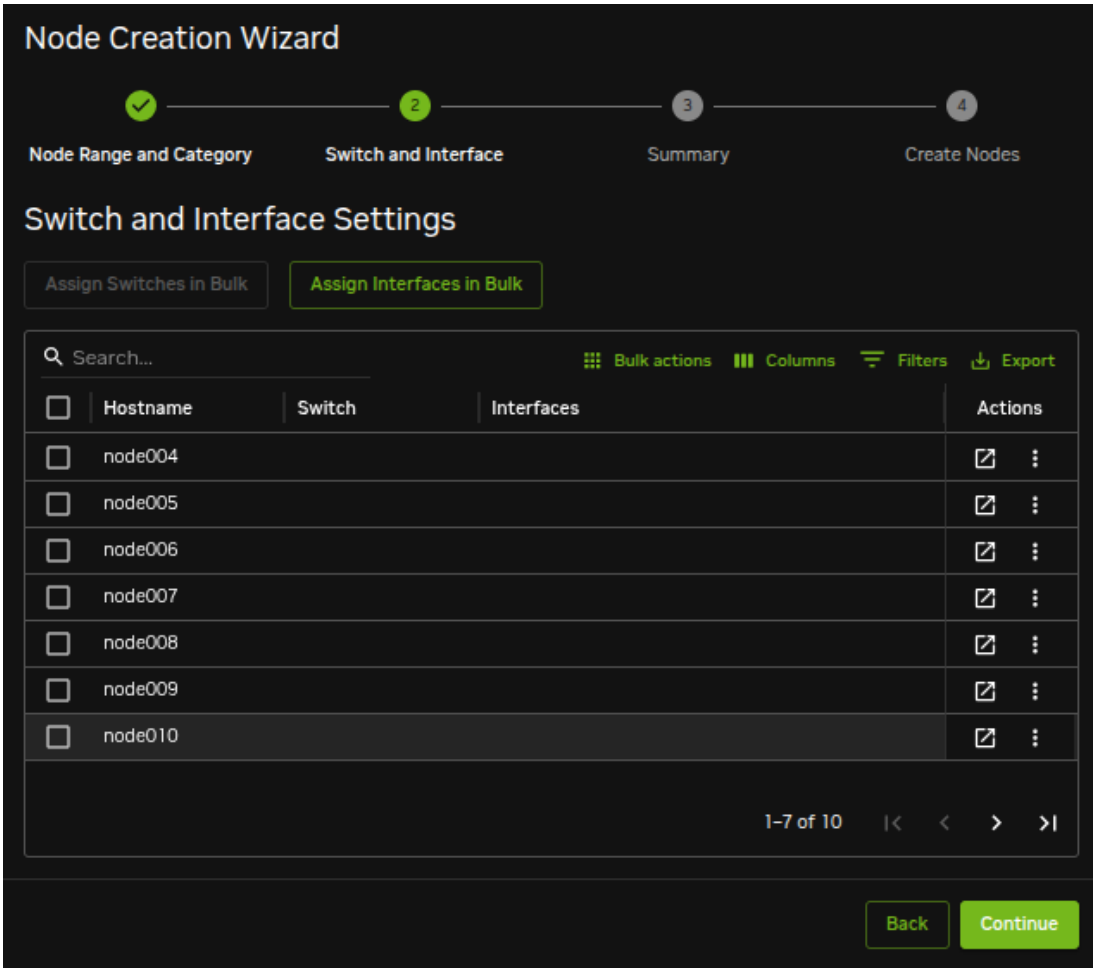


Figure 5.20: Node Creation Wizard: Switch And Interface Screen

The `Assign Interfaces in Bulk` option opens up a subpane that allows an interface to be added to the first node in the list of nodes objects that are to be created (figure 5.21):

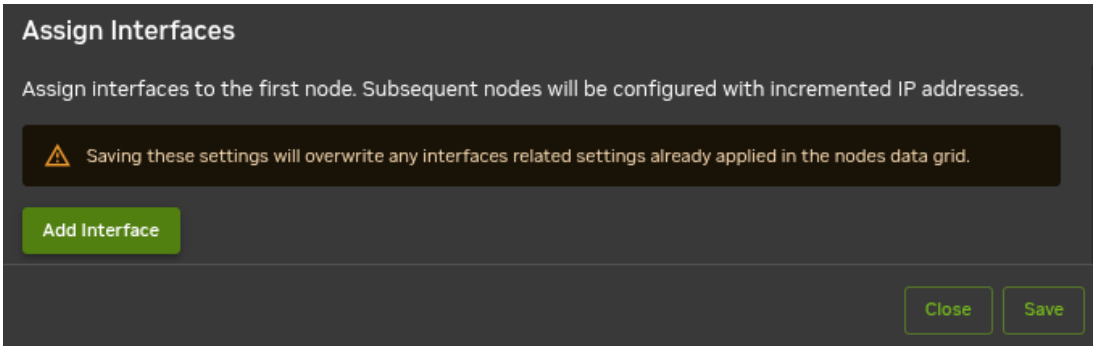


Figure 5.21: Node Creation Wizard: Adding An Interface For Bulk Assignment

In the case of nodes, the interface must be a BOOTIF interface, and should also be set to be used as a provisioning interface. The IP address that is suggested in the screen for the first node should

be changed to a suitable IP address on the management network (figure 5.22):

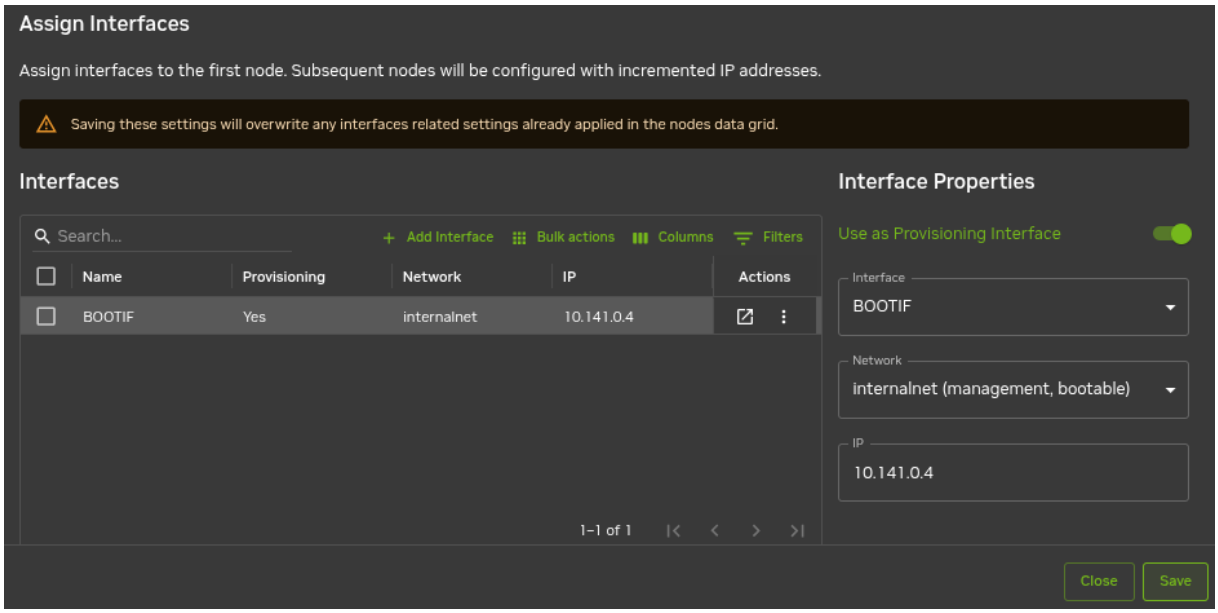


Figure 5.22: Node Creation Wizard: Adding Interface Settings For Bulk Assignment

When the `Save` button is clicked, the remaining nodes in the list of node objects are automatically bulk-assigned IP addresses that follow in a sequence from that of the first node assignment (figure 5.23):

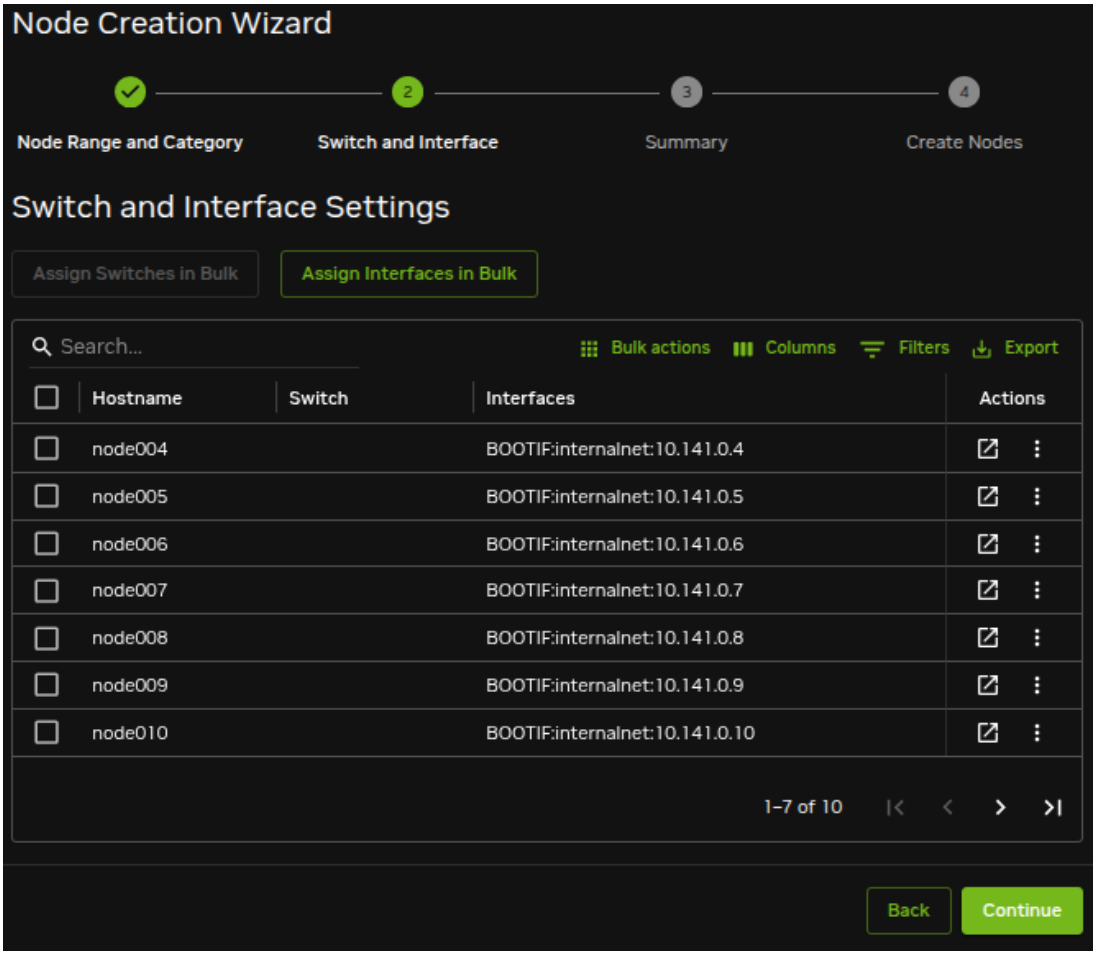


Figure 5.23: Node Creation Wizard: Proposed Bulk Assignment Saved

This assignment is a proposed assignment. Clicking on the `Continue` button brings up the next screen.

- The `Summary` screen: shows a summary (figure 5.24) of the objects that are to be created.

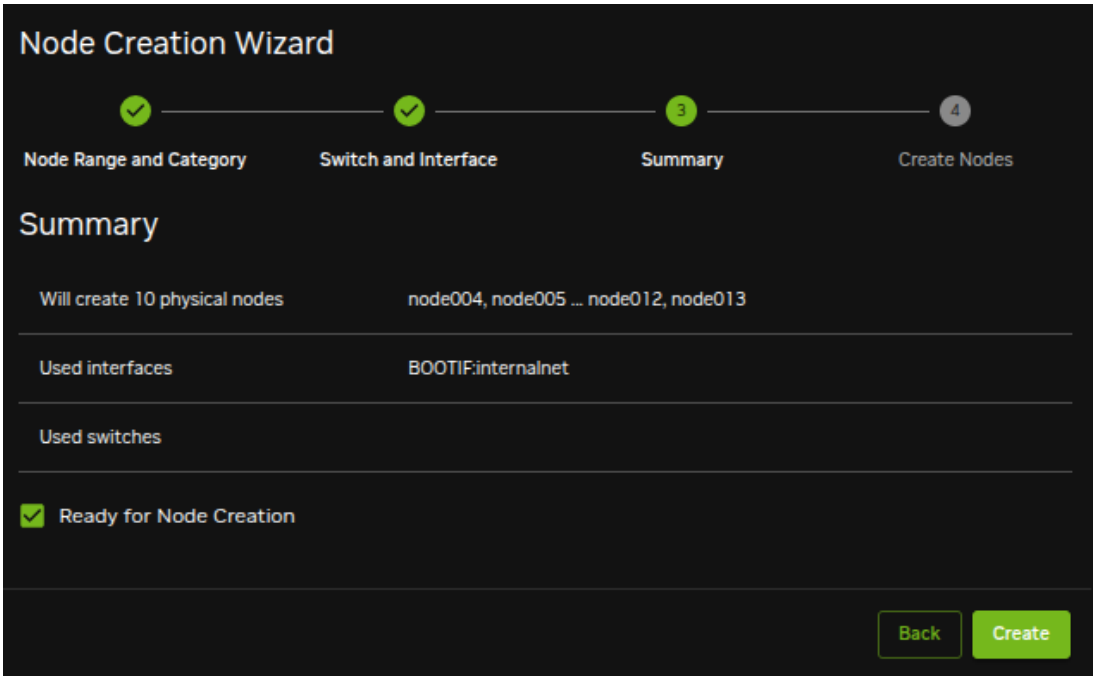


Figure 5.24: Node Creation Wizard: A Summary

After ticking the `Ready for Node Creation` box, clicking on the `Create` button creates the objects.

- The `Create Nodes` screen: shows the status of the node objects (figure 5.25)

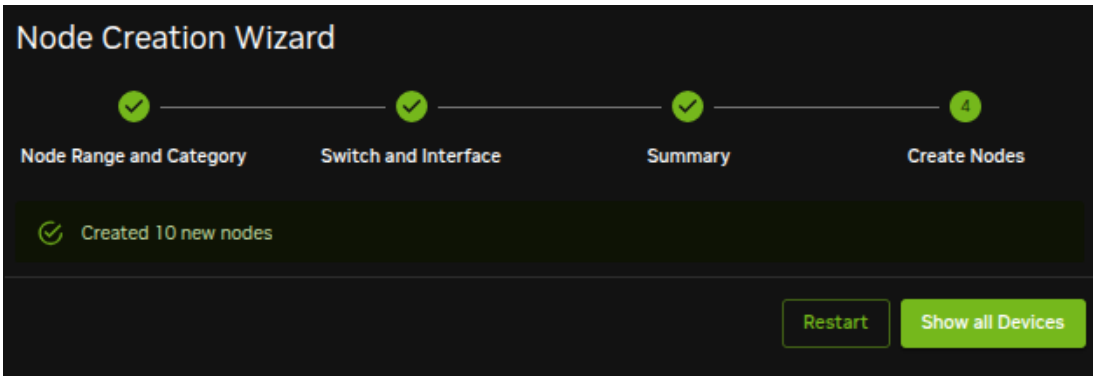


Figure 5.25: Node Creation Wizard: Node Objects Status

The `Show all devices` button just brings up the devices list pane, as seen by the Base View navigation path:

`Devices > All Devices`

The `Restart` button just restarts the wizard to allow more node objects to be created.

Once the node objects have been created, device identification (section 5.4.2) can be carried out as the associated physical nodes are brought up.

The `cmsh` equivalent of the node creation wizard is running `foreach --clone` on a node that is to be cloned over a node range (section 2.5.5, page 73).

5.8 Troubleshooting The Node Boot Process

During the node boot process there are several common issues that can lead to an unsuccessful boot. This section describes these issues and their solutions. It also provides general hints on how to analyze boot problems.

Before looking at the various stages in detail, the administrator may find that simply updating software or firmware may fix the issue. In general, it is recommended that all available updates are deployed on a cluster.

- Updating software is covered in Chapter 9.
  - On the head node, the most relevant software can be updated with `yum`, `zypper`, or `apt`, as explained in section 9.2. For example, with `yum`:

    Example

    ```
    yum update cmdaemon node-installer
    ```

  - Similarly for the software image, the most relevant software can be updated too. This is done via a procedure involving a chroot installation., as described in section 9.4. If using `yum`, then the update can be carried out within the image, <*software image*>, with:

    Example

    ```
    yum update --installroot=/cm/images/<software image> cmdaemon node-installer-slave
    ```

- UEFI or BIOS firmware should be updated as per the vendor recommendation

The various stages that may fail during node boot are now examined.

5.8.1 Node Fails To PXE Boot

Possible reasons to consider if a node is not even starting to network boot (PXE boot for x86 nodes) in the first place:

- DHCP may not be running. A check can be done to confirm that DHCP is running on the internal network interface (usually eth0):

  ```
  [root@basecm11 ~]# ps u -C dhcpd
  USER PID %CPU %MEM VSZ  RSS TTY STAT START TIME COMMAND
  root 2448 0.0 0.0 11208 436 ?   Ss   Jan22 0:05 /usr/sbin/dhcpd eth0
  ```

  This may indicate that `Node booting` is disabled in Base View (figure 3.5, page 105) and needs to be enabled. The equivalent in `cmsh` is to check if the response to:

  ```
  cmsh -c "network use internalnet; get nodebooting"
  ```

  needs to be set to `yes`.

- The DHCP daemon may be "locked down" (section 3.2.1: figure 3.5 and table 3.1). New nodes are granted leases only after `lockdowndhcpd` is set to `no` in `cmsh`, or `Lock down dhcpd` is disabled in Base View for the network.

- A rogue DHCP server may be running. If there are all sorts of other machines on the network the nodes are on, then it is possible that there is a rogue DHCP server active on it, perhaps on an IP address that the administrator has forgotten, and interfering with the expected PXE booting. Such stray DHCP servers should be eliminated.

- One way to identify the problem is to remove all the connections and switches and just connect the head node directly to a problem node, NIC-to-NIC. This should allow a normal network boot to happen. If a normal network boot then does happen, it indicates the problem is indeed due to a rogue DHCP server on the more-connected network.
- For a more cerebral approach, which avoids recabling, the `nmap` utility may be useful. The `nmap` utility since version 7.90 can discover and list multiple DHCP servers using its `broadcast-dhcp-discover` script. The following session output shows the configuration and installation of the utility on to node002 on the internal network. It then runs it for the internal network interface `ens3`. If it finds a second DHCP server on the network (in this test case on node001 at 10.141.0.1), then it may show responses in the output similar to the following (some output ellipsized):

Example

```
[root@node002 ~]# wget https://nmap.org/dist/nmap-7.90.tgz
...
[root@node002 ~]# tar xvzf nmap-7.90
...
[root@node002 ~]# cd nmap-7.90
[root@node002 nmap-7.90]# make distclean && ./configure --disable-rdma && make
...
[root@node002 nmap-7.90]# ./nmap --script broadcast-dhcp-discover -e ens3
Starting Nmap 7.90 ( https://nmap.org ) at 2022-09-08 16:55 CEST
Pre-scan script results:
| broadcast-dhcp-discover:
|   Response 1 of 2:
|     IP Offered: 10.141.163.254
|     DHCP Message Type: DHCPOFFER
|     Server Identifier: 10.141.0.1
...
|   Response 2 of 2:
|     IP Offered: 10.141.167.255
|     DHCP Message Type: DHCPOFFER
|     Server Identifier: 10.141.255.254
...
[root@node002 nmap-7.90]#
```

- The boot sequence may be set wrongly in the BIOS. The boot interface should normally be set to be the first boot item in the BIOS.
- The node may be set to boot from UEFI mode. If UEFI mode has a buggy network boot implementation, then it may fail to network boot. For x86 nodes, setting the node to PXE boot using the legacy BIOS mode can be tried instead, or perhaps the UEFI firmware can be updated.
- There may a bad cable connection. This can be due to moving the machine, or heat creep, or another physical connection problem. Firmly inserting the cable into its slot may help. Replacing the cable or interface as appropriate may be required.
- There may a problem with the switch. Removing the switch and connecting a head node and a regular node directly with a cable can help troubleshoot this.

  Disabling the Spanning Tree Protocol (STP) functions of a managed switch is recommended. With STP on, nodes may randomly fail to network boot.
- The cable may be connected to the wrong interface. By default, on the head node, for a type 1 network, the first consistent network device name, for example `eno1`, is normally assigned the

internal network interface, and the second one, for example en02, is assigned the external network interface. However, the following possibilities should be considered during troubleshooting:

- The two interfaces can be confused when physically viewing them and a connection to the wrong interface can therefore be made.
- It is also possible that the administrator has changed the default assignment.
- The interface may have been set by the administrator to follow the network device naming scheme that has been used prior to RHEL7. Interfaces with names such as eth0 and eth1 on the head node are suggestive of this. The problem with the pre-RHEL7 scheme is that it can sometimes lead to network interfaces swapping after reboot, which is why the scheme is no longer recommended. The workaround for this issue in pre-RHEL7 schemes was to define a persistent name in the udev ruleset for network interfaces.

  From NVIDIA Base Command Manager version 9.0 onward, the default scheme is the consistent network device naming scheme, and it is recommended.

Interface Naming Conventions Post-RHEL7 (Recommended)

https://access.redhat.com/documentation/en-US/Red_Hat_Enterprise_Linux/7/html/Networking_Guide/ch-Consistent_Network_Device_Naming.html describes the consistent network device scheme for interfaces post-RHEL7. This scheme sets an interface assignment on iPXE boot for multiple interfaces that is also valid by default during the very first iPXE boot. This means that an administrator can know which interface is used for provisioning and can connect the provisioning cable accordingly.

Some care may need to be taken in unusual naming assignments, in order to avoid exceeding the 16-character limit that Linux has for the naming of network interfaces.

Reverting To The Pre-RHEL7 Interface Naming Conventions (Not Recommended)

To revert to the pre-RHEL7 behavior, the key=value entries:

```
net.ifnames=0 biosdevname=0
```

can be placed in the GRUB_CMDLINE_LINUX line in /etc/default/grub within the head node:

Example

```
GRUB_CMDLINE_LINUX="vconsole.keymap=us crashkernel=auto vconsole.font=latarcyrheb-sun16 \
biosdevname=0 rootdelay=300 net.ifnames=0 earlyprintk=ttyS0,115200 console=ttyS0"
```

For this:

* The biosdevname parameter only works if the dev helper is installed. The dev helper is available from the biosdevname RPM package. The parameter also requires that the system supports SMBIOS 2.6 or ACPI DSM.
* The net.ifnames parameter is needed if biosdevname is not installed.

A cautious system administrator may back up the original grub.cfg file:

```
[root@basecm11 ~]# cp --preserve /boot/grub2/grub.cfg /boot/grub2/grub.cfg.orig
```

The GRUB configuration should be generated with:

```
[root@basecm11 ~]# grub2-mkconfig -o /boot/grub2/grub.cfg
```

If for some reason the administrator would like to carry out the pre-RHEL7 naming convention on a regular node, then the text net.ifnames=0 biosdevname=0 can be appended to the kernelparameters property, for an image selected from softwareimage mode.

Example

```
[basecm11->softwareimage]% list
Name (key)           Path
-------------------- ------------------------------
default-image        /cm/images/default-image
openstack-image      /cm/images/openstack-image
[basecm11->softwareimage]% use default-image
[basecm11->softwareimage[default-image]]% append kernelparameters " net.ifnames=0 biosdevname=0"
[basecm11->softwareimage*[default-image*]]% commit
```

The append command requires a space at the start of the quote, in order to separate the kernel parameters from any pre-existing ones.

- The TFTP server that sends out the image may have hung. During a normal run, an output similar to this appears when an image is in the process of being served:

```
[root@basecm11 ~]# ps ax | grep [t]ftp
 7512 ?        Ss     0:03 in.tftpd --maxthread 500 /tftpboot
```

If the TFTP server is in a zombie state, the head node should be rebooted. If the TFTP service hangs regularly, there is likely a networking hardware issue that requires resolution.

Incidentally, grepping the process list for a TFTP service returns nothing when the head node is listening for TFTP requests, but not actively serving a TFTP image. This is because the TFTP service runs under xinet.d and is called on demand. Running

```
[root@basecm11 ~]# chkconfig --list
```

should include in its output the line:

```
tftp:     on
```

if TFTP is running under xinet.d.

- The switchover process from TFTP to HTTP may have hung. During a normal provisioning run, assuming that CMDaemon uses the default `bootloaderprotocol` setting of `HTTP`, then TFTP is used to load the initial boot loader, but the kernel and ramdisk are loaded up via HTTP for speed. Some hardware has problems with switching over to using HTTP.

  In that case, setting `bootloaderprotocol` to `TFTP` keeps the node using TFTP for loading the kernel and ramdisk, and should work. Another possible way to solve this is to upgrade the PXE boot BIOS to a version that does not have this problem.

  ARMv8 hardware can boot only via `TFTP`.

  Setting `bootloaderprotocol` to `HTTPS` only works for some special hardware.

- VLAN tagging may have been set up incorrectly in the BIOS of the node. VLAN provisioning (section 5.3.4) requires several changes in VLAN configuration for it to work.

- Sometimes a manufacturer releases hardware with buggy drivers that have a variety of problems. For instance: Ethernet frames may be detected at the interface (for example, by `ethtool`), but TCP/IP packets may not be detected (for example, by `wireshark`). In that case, the manufacturer should be contacted to upgrade their driver.

- The interface may have a hardware failure. In that case, the interface should be replaced.

5.8.2 Node-installer Logging

If the node manages to get beyond the net booting stage to the node-installer stage, then the first place to look for hints on node boot failure is usually the node-installer log file. The node-installer runs on the node that is being provisioned, and sends logging output to the syslog daemon running on that node. This forwards all log data to the IP address from which the node received its DHCP lease, which is typically the IP address of the head node or failover node. In a default BCM setup, the `local5` facility of the `syslog` daemon is used on the node that is being provisioned to forward all node-installer messages to the log file `/var/log/node-installer` on the head node.

After the node-installer has finished running, its log is also stored in `/var/log/node-installer` on the regular nodes.

If there is no node-installer log file anywhere yet, then it is possible that the node-installer is not yet deployed on the node. Sometimes this is due to a system administrator having forgotten to change a provisioning-related configuration setting. One possibility is that the `nodegroups` setting (section 5.2.1), if used, may be misconfigured. Another possibility is that the image was set to a `locked` state (section 5.4.7). The `provisioningstatus -a` command can indicate this:

Example

```
[basecm11->softwareimage]% provisioningstatus -a | grep locked
Scheduler info:  requested software image is locked, request deferred
```

To get the image to install properly, the `locked` state should be removed for a locked image.

Example

```
[root@basecm11 ~]# cmsh -c "softwareimage islocked"
Name           Locked
-------------- --------
default-image  yes
[root@basecm11 ~]# cmsh -c "softwareimage unlock default-image"
[root@basecm11 ~]# cmsh -c "softwareimage islocked"
Name           Locked
-------------- --------
default-image  no
```

The node automatically picks up the image after it is unlocked.

Optionally, extra log information can be written by enabling debug logging, which sets the syslog importance level at `LOG_DEBUG`. To enable debug logging, the debug field is changed in `/cm/node-installer/scripts/node-installer.conf`.

For the `node-installer.conf` file in multidistro and multiarch (section 9.7) configurations, the directory path `/cm/node-installer` takes the form:

`/cm/node-installer-`<*distribution*>`-`<*architecture*>

The values for <*distribution*> and <*architecture*> can take the values outlined on page 558.

From the console of the booting node the log file is generally accessible by pressing `Alt+F7` on the keyboard. Debug logging is however excluded from being viewed in this way, due to the output volume making this impractical.

A booting node console can be accessed remotely if Serial Over LAN (SOL) is enabled (section 14.7), to allow the viewing of console messages directly. A further depth in logging can be achieved by setting the kernel option `loglevel=N`, where `N` is a number from `0` (`KERN_EMERG`) to `7` (`KERN_DEBUG`).

One possible point at which the node-installer can fail on some hardware is if SOL (section 14.7) is enabled in the BIOS, but the hardware is unable to cope with the flow. The installation can freeze completely at that point. This should not be confused with the viewing quirk described in section 14.7.4, even though the freeze typically appears to take place at the same point, that point being when the console shows "`freeing unused kernel memory`" as the last text. One workaround to the freeze would be to disable SOL.

5.8.3 Provisioning Logging

The provisioning system sends log information to the CMDaemon log file. By default this is in `/var/log/cmdaemon` on the local host, that is, the provisioning host. The host this log runs on can be configured with the CMDaemon directive `SyslogHost` (Appendix C).

The image synchronization log file can be retrieved with the `synclog` command (page 285) running from device mode in `cmsh`. Hints on provisioning problems are often found by looking at the tail end of the log.

If the tail end of the log shows an rsync exit code of 23, then it suggests a transfer error. Sometimes the cause of the error can be determined by examining the file or filesystem for which the error occurs. For example, the following `synclog` output:

Example

```
[...]
sending incremental file list
rsync: [generator] delete_file: rmdir(lscratch) failed: Device or resource busy (16)
[...]
```

could be due to a missing `lscratch` mountpoint in the software image.

For the rsync transport, logs for node installation are kept under `/var/spool/cmd/rsync/`, with a log written for each node during provisioning. The name of the node is set as the prefix to the log name. For example `node002` generates the log:

```
/var/spool/cmd/rsync/node002-\.log
```

5.8.4 Ramdisk Fails During Loading Or Sometime Later

One issue that may come up after a software image update via `yum`, `zypper`, or `apt` (section 9.4), is that the ramdisk stage may fail during loading or sometime later, for a node that is rebooted after the update. This occurs if there are instructions to modify the ramdisk by the update. In a normal machine the ramdisk would be regenerated. In a cluster, the extended ramdisk that is used requires an update, but BCM is not aware of this.

Updating the RAM disk for the cluster solves the failure for this case. This can be done:

- by running the `createramdisk` command from `cmsh` (section 5.3.2), using the appropriate modes (`device`, `category`, or `software image`), or
- by running the `Create RAM Disk` command using the appropriate Base View navigation paths:
  - `Devices > All Devices >` *selected node*`: > Kernel > Create RAM Disk` (typically inherited from software image by default)
  - `Grouping > Categories >` *selected category*`: > Kernel > Create RAM Disk` (typically blank by default)
  - `Provisioning > Software Images >` *selected software image*`: > Kernel > Create RAM Disk`

Another, somewhat related possible cause of a halt at this stage, is that the kernel modules that are to be loaded may have been specified incorrectly by the administrator in the hierarchy of software image, category, or node (page 256). A check of the kernel modules specified in `softwareimage` mode, `category` mode, or `device` mode (for the particular node) may reveal a misconfiguration.

5.8.5 Ramdisk Fails With `No space left on device`

The `createramdisk` log for a software image may show a message:

```
... abort: write error on <stdout> (No space left on device)
```

This error can be caused due to a large ramdisk being generated that exceeds the default tmpfs size configured by CMDaemon.

To solve this, a larger default tmpfs size value, in MB, can be specified with an `AdvancedConfig` (page 925) directive in `cmd.conf`, on all the head nodes in the cluster:

Example

```
AdvancedConfig = { "RamdiskCreationTmpfsSize=512" }
```

After increasing the value with an `AdvancedConfig` directive, a restart of CMDaemon service is required on all head nodes for the change to take effect.

The default tmpfs size was 256MB for BCM 11 versions before 11.31.0. Due to kernel growth, this was sometimes inadequate. The default size was therefore set to 512MB for BCM 11 versions from 11.31.0 onward.

5.8.6 Ramdisk Cannot Start Network

The ramdisk must activate the node's network interface in order to fetch the node-installer. To activate the network device, the correct kernel module needs to be loaded. If this does not happen, booting fails, and the console of the node displays something similar to figure 5.26.

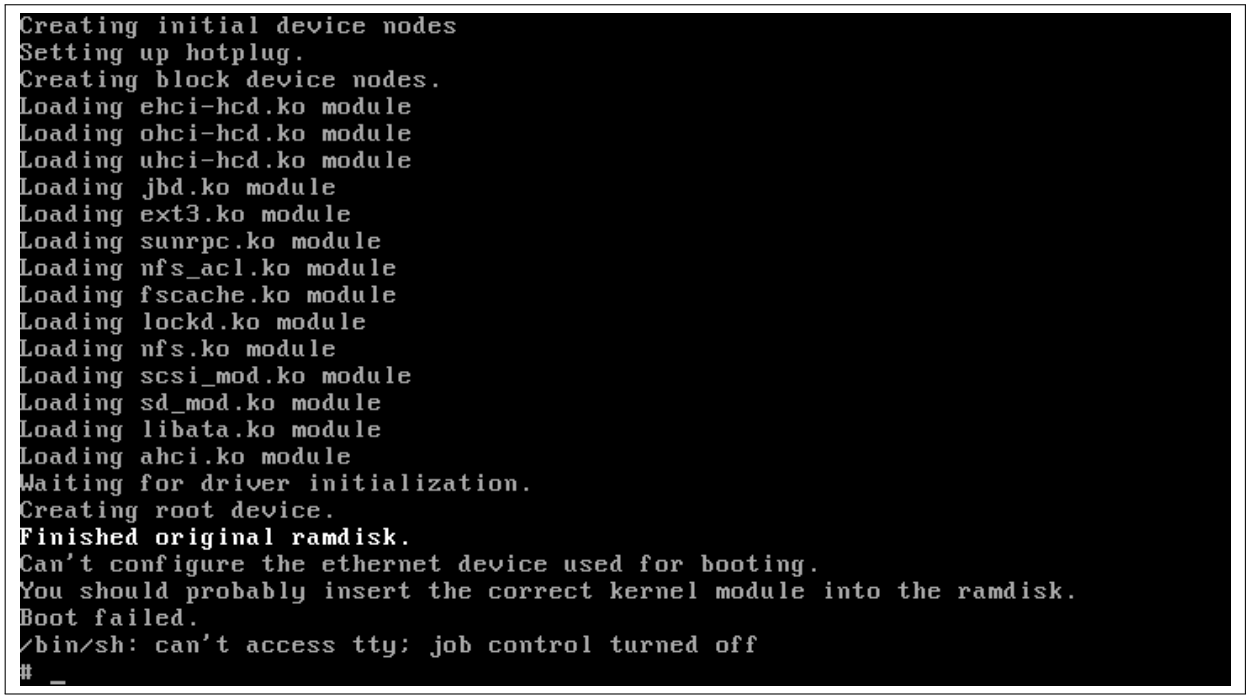


Figure 5.26: No Network Interface

To solve this issue the correct kernel module should be added to the software image's kernel module configuration (section 5.3.2). For example, to add the e1000 module to the default image using `cmsh`:

Example

```
[mc]% softwareimage use default-image
[mc->softwareimage[default-image]]% kernelmodules
[mc->softwareimage[default-image]->kernelmodules]% add e1000
[mc->softwareimage[default-image]->kernelmodules[e1000]]% commit
Initial ramdisk for image default-image was regenerated successfully
[mc->softwareimage[default-image]->kernelmodules[e1000]]%
```

After committing the change it typically takes about a minute before the initial ramdisk creation is completed via a `mkinitrd` run by CMDaemon.

5.8.7 Node-Installer Cannot Create Disk Layout

When the node-installer is not able to create a drive layout it displays a message similar to figure 5.27. The node-installer log file (section 5.8.2) contains something like:

```
Mar 24 13:55:31 10.141.0.1 node-installer: Installmode is: AUTO
Mar 24 13:55:31 10.141.0.1 node-installer: Fetching disks setup.
Mar 24 13:55:31 10.141.0.1 node-installer: Checking partitions and
filesystems.
Mar 24 13:55:32 10.141.0.1 node-installer: Detecting device '/dev/sda':
not found
Mar 24 13:55:32 10.141.0.1 node-installer: Detecting device '/dev/hda':
not found
Mar 24 13:55:32 10.141.0.1 node-installer: Can not find device(s) (/dev/sda /dev/hda).
Mar 24 13:55:32 10.141.0.1 node-installer: Partitions and/or filesystems
are missing/corrupt. (Exit code 4, signal 0)
Mar 24 13:55:32 10.141.0.1 node-installer: Creating new disk layout.
Mar 24 13:55:32 10.141.0.1 node-installer: Detecting device '/dev/sda':
not found
Mar 24 13:55:32 10.141.0.1 node-installer: Detecting device '/dev/hda':
not found
Mar 24 13:55:32 10.141.0.1 node-installer: Can not find device(s) (/dev/sda /dev/hda).
Mar 24 13:55:32 10.141.0.1 node-installer: Failed to create disk layout.
(Exit code 4, signal 0)
Mar 24 13:55:32 10.141.0.1 node-installer: There was a fatal problem. This node can not be\
 installed until the problem is corrected.
```

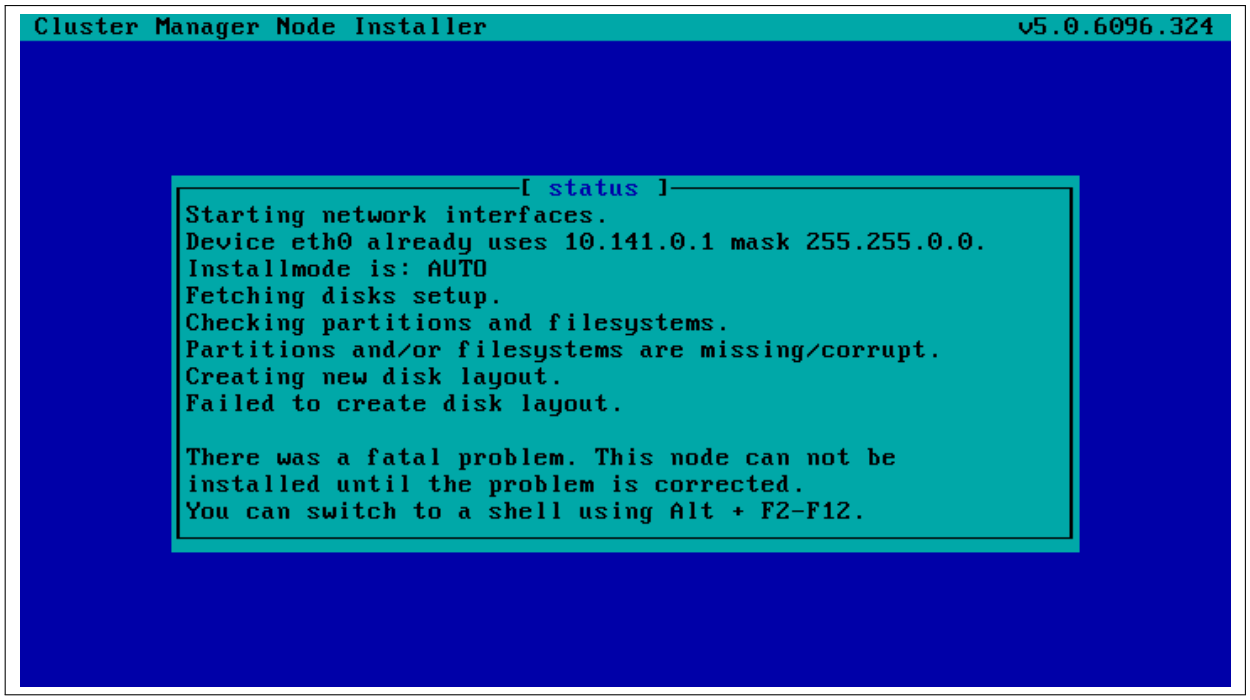


Figure 5.27: No Disk

Disk layout failures can have several reasons.

BIOS And Order Issues

One reason may be that the drive may be disabled in the BIOS. It should be enabled.

Another reason may be that the drive order was changed. This could happen if, for example, a defective motherboard has been replaced. The drive order should be kept the same as it was before a motherboard change.

Read-only Mode Issues

Another reason may be due to SSDs that have a hardware jumper or toggle switch that sets a drive to read-only mode. A read-only mode drive will typically fail at this point. The drive should be made writeable.

Hardware Issues

If the node-installer log for the node shows lines with the text `Input/output error`, then it generally indicates a hardware issue. Possible hardware issues include:

- a drive failure
- a faulty cable between storage and controller
- a faulty storage controller
- a faulty backplane in the server

If the node has enough RAM, then it is possible to boot up the node as a diskless node, to carry out further diagnosis with disk tools such as `smartmontools`. The procedure for changing a node's disk layout to a diskless configuration is given in section 3.13.

Automatic Diskless Fallback On Missing Device

The RAM-based diskless boot mentioned in the preceding text requires an administrator to decide to configure a node to install by bypassing a disk. This could be carried out for a node that has a functioning

disk or it could be carried out for a node with a non-functioning disk.

A RAM-based diskless boot can be carried out with the decision to bypass made automatically instead, by setting the `switchToDisklessOnMissingDevice` directive to `true` in the `node-installer.conf` file. The node-installer configuration file is located at `/cm/node-installer/scripts/node-installer.conf` (at `/cm/node-installer-<distribution>-<architecture>/scripts/node-installer.conf` for multi-arch/multidistro configurations).

If `switchToDisklessOnMissingDevice` is set to `true`, then a missing drive no longer causes a fatal error. The node-installer instead falls back to a diskless installation scheme, and the node boots up on its own. Further diagnosis can then be carried out on a live system, for example with `smartmontools`, or—on a DGX node—an `nvsm` dump. The default value is `false`, which preserves the original behavior of refusing to install the node when an expected drive cannot be found.

Software Driver Issues

One of the most common software issues is that the correct storage driver is not being loaded. To solve this issue, the correct kernel module should be added to the software image's kernel module configuration (section 5.3.2).

Experienced system administrators work out what drivers may be missing by checking the results of hardware probes. For example, going into the node-installer shell using `Alt-F2`, and then looking at the output of `lspci`, shows a list of hardware detected in the PCI slots and gives the chipset name of the storage controller hardware in this case:

Example

```
[<installer> root@node001 ~]# lspci | grep SCSI
00:10.0 Serial Attached SCSI controller: LSI Logic / Symbios Logic SAS2\
008 PCI-Express Fusion-MPT SAS-2 [Falcon] (rev 03)
```

The next step is to Google with likely search strings based on that output.

The Linux Kernel Driver DataBase (LKDDb) is a hardware database built from kernel sources that lists driver availability for Linux. It is available at `http://cateee.net/lkddb/`. Using the Google search engine's "`site`" operator to restrict results to the `cateee.net` web site only, a likely string to try might be:

Example

```
SAS2008 site:cateee.net
```

The search result indicates that the `mpt2sas` kernel module needs to be added to the node kernels. A look in the modules directory of the software image shows if it is available:

Example

```
find /cm/images/default-image/lib/modules/ -name "*mpt2sas*"
```

If it is not available, the driver module must then be obtained. If it is a source file, it will need to be compiled. By default, nodes run on standard distribution kernels, so that only standard procedures need to be followed to compile modules.

If the module is available, it can be added to the default image, by using `cmsh` in `softwareimage` mode to create the associated object. The object is given the same name as the module, i.e. `mp2sas` in this case:

Example

```
[basecm11]% softwareimage use default-image
[basecm11->softwareimage[default-image]]% kernelmodules
[basecm11->softwareimage[default-image]->kernelmodules]% add mpt2sas
[basecm11->softwareimage[default-image]->kernelmodules*[mpt2sas*]]% commit
[basecm11->softwareimage[default-image]->kernelmodules[mpt2sas]]%
Thu May 19 16:54:52 2011 [notice] basecm11: Initial ramdisk for image de\
fault-image is being generated
[basecm11->softwareimage[default-image]->kernelmodules[mpt2sas]]%
Thu May 19 16:55:43 2011 [notice] basecm11: Initial ramdisk for image de\
fault-image was regenerated successfully.
[basecm11->softwareimage[default-image]->kernelmodules[mpt2sas]]%
```

After committing the change it can take some time before ramdisk creation is completed—typically about a minute, as the example shows. Once the ramdisk is created, the module can be seen in the list displayed from `kernelmodules` mode. On rebooting the node, it should now continue past the disk layout stage.

5.8.8 Node-Installer Cannot Start BMC (IPMI/iLO) Interface

In some cases the node-installer is not able to configure a node's BMC interface, and displays an error message similar to figure 5.28.

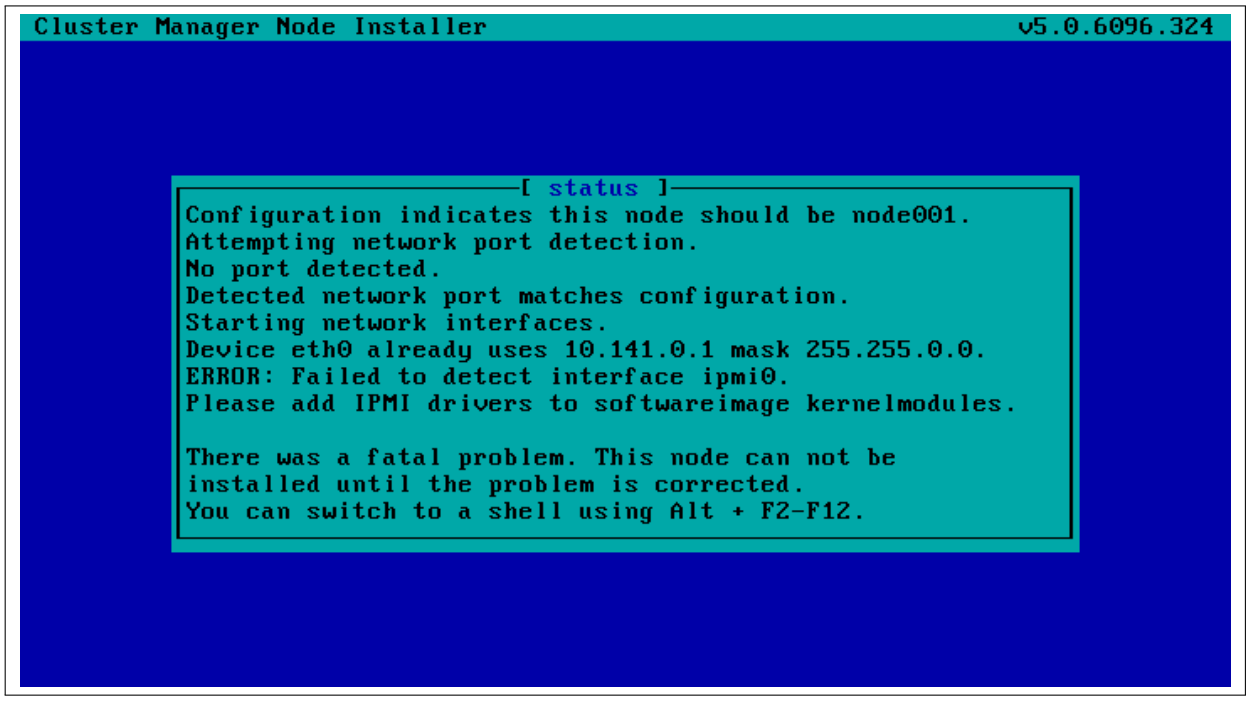


Figure 5.28: No BMC Interface

Troubleshooting may involve trying out node identification and port queries with a BMC (section 3.7.5), as well as other steps that are described in the text that follows.

Often the issue can be solved by adding the correct BMC (IPMI/iLO) kernel modules to the software image's kernel module configuration. However, in some cases the node-installer is still not able to configure the BMC interface. If this is the case then the BMC probably does not support one of the commands the node-installer uses to set specific settings, or there may be a hardware glitch in the BMC.

The setupBmc Node-Installer Configuration Setting

To solve this issue, setting up BMC interfaces can be disabled globally by setting the setupBmc field to false in the node-installer configuration file /cm/node-installer/scripts/node-installer.conf (for multiarch/multidistro configurations the path takes the form: /cm/node-installer-<*distribution*>-<*architecture*>/scripts/node-installer.conf).

Doing this disables configuration of all BMC interfaces by the node-installer. A custom finalize script (Appendix E) can then be used to run the required commands instead.

The setupBmc field in the node-installer should not be confused with the SetupBMC directive in cmd.conf (page 918, Appendix C). The former is about enabling the BMC interface, while the latter is about enabling automated passwords to the BMC interface (an interface that must of course be enabled in the first place to work).

The failOnMissingBmc Node-Installer Configuration Setting

If the kernel modules for the BMC are loaded up correctly, and the BMC is configured, but it is not detected by the node-installer, then the node-installer halts by default. This corresponds to the setting failOnMissingBmc = true in the node-installer configuration file /cm/node-installer/scripts/node-installer.conf. Toggling this to false skips BMC network device detection, and lets the node-installer continue past the BMC detection and configuration stage. This can be convenient, for example, if the BMC is not yet configured and the aim is to get on with setting up the rest of the cluster.

The failOnFailedBmcCommand Node-Installer Configuration Setting

If a BMC command fails, then the node-installer by default terminates node installation. The idea behind this is to allow the administrator to fix the problem. Sometimes, however, hardware can wrongly signal a failure. That is, it can signal a false failure, as opposed to a true failure.

A common case is the case of ipmitool. ipmitool is used by BCM to configure the BMC. With most hardware vendors it works as expected, signaling success and failure correctly. As per the default behavior: with success, node installation proceeds, while with failure, it terminates.

With certain hardware vendors however ipmitool fails with an exit code 1, even though the BMC is properly configured. Again, as per the default behavior: success has node installation proceed, while failure has node installation terminate. Only this time, because the failure signal is incorrect, the termination on failure is also incorrect behavior.

To get around the default behavior for false failure cases, the administrator can force the node-installer to set the value of failOnFailedBmcCommand to false in the node-installer configuration file /cm/node-installer/scripts/node-installer.conf (for multiarch/multidistro configurations the path takes the form: /cm/node-installer-<*distribution*>-<*architecture*>/scripts/node-installer.conf). The installation then skips past the false failure.

BMC Hardware Glitch And Cold Reset

Sometimes, typically due to a hardware glitch, a BMC can get into a state where it is not providing services, but the BMC is still up (responding to pings). Contrariwise, a BMC may not respond to pings, but still respond to IPMI commands. A fix for such glitchy states is usually to power cycle the BMC. This is typically done, either physically, or by using a BMC management tool such as ipmitool.

Physically resetting the power supply to the BMC is done typically by pulling the power cable out and then pushing it in again. For typical rack-based servers the server can just be pulled out and in again. Just doing a shutdown of the server with the power cable still in place normally does not power down the BMC.

BMC management does allow the BMC to power down and be reset from software, without having to physically handle the server. This software-based *cold reset* is a BIOS-manufacturer-dependent feature. A popular tool used for managing BMCs that can do such a cold reset is ipmitool. This can be run remotely, but also on the node console if the node cannot be reached remotely.

With ipmitool, a cold reset is typically carried out with a command such as:

```
[root@basecm11 ~]# module load ipmitool
```

```
[root@basecm11 ~]# ipmitool -U <bmcusername> -P <bmcpassword> -H <host IP> -I lanplus mc reset cold
```

The values for <*bmcusername*> and <*bmcpassword*> can be obtained as shown in section 3.7.2.

BMC Troubleshooting With The System Event Log

The System Event Log (SEL) can be read with:

```
[root@basecm11 ~]# module load ipmitool
[root@basecm11 ~]# ipmitool -U <bmcusername> -P <bmcpassword> -H <host IP> -I lanplus sel list
```

The timestamped output can be inspected for errors related to the CPU, ECC, or memory.

Other BMC Troubleshooting

Some more specific commands for handling IPMI might be using the `service ipmi <option>` commands, which can show the IPMI service has failed to start up:

Example

```
[root@basecm11 ~]# service ipmi status
Redirecting to /bin/systemctl status  ipmi.service
   ipmi.service - IPMI Driver
   Loaded: loaded (/usr/lib/systemd/system/ipmi.service; disabled; vendor preset: enabled)
   Active: inactive (dead)
```

In the preceding session the driver has simply not been started up. It can be started up with the `start` option:

Example

```
[root@basecm11 ~]# service ipmi start
Redirecting to /bin/systemctl start  ipmi.service
Job for ipmi.service failed because the control process exited with error code. See "systemctl\
 status ipmi.service" and "journalctl -xe" for details.
```

In the preceding session, the start up failed. The service status output shows:

Example

```
[root@basecm11 ~]# service ipmi status -l
Redirecting to /bin/systemctl status  -l ipmi.service
   ipmi.service - IPMI Driver
   Loaded: loaded (/usr/lib/systemd/system/ipmi.service; disabled; vendor preset: enabled)
   Active: failed (Result: exit-code) since Mon 2016-12-19 14:34:27 CET; 2min 3s ago
  Process: 8930 ExecStart=/usr/libexec/openipmi-helper start (code=exited, status=1/FAILURE)
 Main PID: 8930 (code=exited, status=1/FAILURE)

Dec 19 14:34:27 basecm11 systemd[1]: Starting IPMI Driver...
Dec 19 14:34:27 basecm11 openipmi-helper[8930]: Startup failed.
Dec 19 14:34:27 basecm11 systemd[1]: ipmi.service: main process exited, code=exited, status=1/\
FAILURE
Dec 19 14:34:27 basecm11 systemd[1]: Failed to start IPMI Driver.
Dec 19 14:34:27 basecm11 systemd[1]: Unit ipmi.service entered failed state.
Dec 19 14:34:27 basecm11 systemd[1]: ipmi.service failed.
```

Further details can be found in the journal:

Example

```
[root@basecm11 ~]# journalctl -xe | grep -i ipmi
...
-- Unit ipmi.service has begun starting up.
Dec 19 14:34:27 basecm11 kernel: ipmi message handler version 39.2
Dec 19 14:34:27 basecm11 kernel: IPMI System Interface driver.
Dec 19 14:34:27 basecm11 kernel: ipmi_si: Unable to find any System Interface(s)
Dec 19 14:34:27 basecm11 openipmi-helper[8930]: Startup failed.
...
```

In the preceding session, the failure is due to a missing BMC interface (`Unable to find any System Interface(s)`). A configured BMC interface should show an output status similar to:

Example

```
[root@basecm11 ~]# service ipmi status
Redirecting to /bin/systemctl status  ipmi.service
   ipmi.service - IPMI Driver
   Loaded: loaded (/usr/lib/systemd/system/ipmi.service; disabled; vendor preset: enabled)
   Active: active (exited) since Mon 2016-12-19 14:37:10 CET; 2min 0s ago
  Process: 61019 ExecStart=/usr/libexec/openipmi-helper start (code=exited, status=0/SUCCESS)
 Main PID: 61019 (code=exited, status=0/SUCCESS)
Dec 19 14:37:10 basecm11 systemd[1]: Starting IPMI Driver...
Dec 19 14:37:10 basecm11 systemd[1]: Started IPMI Driver.
```

Sometimes the issue may be an incorrect networking specification for the BMC interfaces. MAC and IP details that have been set for the BMC interface can be viewed with the `lan print` option to `ipmitool` if the service has been started:

Example

```
[root@basecm11 ~]# module load ipmitool
[root@basecm11 ~]# ipmitool lan print
Set in Progress         : Set Complete
Auth Type Support       : MD5 PASSWORD
Auth Type Enable        : Callback : MD5 PASSWORD
                        : User     : MD5 PASSWORD
                        : Operator : MD5 PASSWORD
                        : Admin    : MD5 PASSWORD
                        : OEM      :
IP Address Source       : Static Address
IP Address              : 93.184.216.34
Subnet Mask             : 255.255.255.0
MAC Address             : aa:bb:01:02:cd:ef
SNMP Community String   : public
IP Header               : TTL=0x00 Flags=0x00 Precedence=0x00 TOS=0x00
BMC ARP Control         : ARP Responses Enabled, Gratuitous ARP Disabled
Gratituous ARP Intrvl   : 0.0 seconds
Default Gateway IP      : 93.184.216.1
Default Gateway MAC     : 00:00:00:00:00:00
Backup Gateway IP       : 0.0.0.0
Backup Gateway MAC      : 00:00:00:00:00:00
802.1q VLAN ID          : Disabled
802.1q VLAN Priority    : 0
RMCP+ Cipher Suites     : 0,1,2,3,4,6,7,8,9,11,12,13,15,16,17,18
Cipher Suite Priv Max   : caaaaaaaaaaaaaa
                        :     X=Cipher Suite Unused
                        :     c=CALLBACK
```

```
:     u=USER
:     o=OPERATOR
:     a=ADMIN
:     O=OEM
```

During normal operation the metrics (Appendix G) displayed by BCM are useful. However, if those are not available for some reason, then the direct output from BMC sensor metrics may be helpful for troubleshooting:

Example

```
[root@basecm11 ~]# module load ipmitool
[root@basecm11 ~]# ipmitool sensor list all
# ipmitool sensor list
Ambient Temp | 22.000   | degrees C | ok | na | na       | na | 38.000 | 41.000 | 45.000
AVG Power    | 300.000  | Watts     | ok | na | na       | na | na     | na     | na
Fan 1 Tach   | 4125.000 | RPM       | ok | na | 750.000 | na | na     | na     | na
...
```

6

User Management

Users and groups for the cluster are presented to the administrator in a single system paradigm. That is, if the administrator manages them with BCM, then the changes are automatically shared across the cluster (the single system).

BCM runs its own LDAP service to manage users, rather than using unix user and group files. In other words, users and groups are managed via the centralizing LDAP database server running on the head node, and not via entries in `/etc/passwd` or `/etc/group` files.

Sections 6.1 and 6.2 cover the most basic aspects of how to add, remove and edit users and groups using BCM.

Section 6.3 describes how an external LDAP server can be used for authentication services instead of the one provided by BCM.

Section 6.5 discusses how users can be assigned only selected capabilities when using Base View or `cmsh`, using profiles with sets of tokens. For example, a regular BCM user may be assigned the set of tokens associated with the `admin` profile in order to avoid using the `root` account for BCM administration.

6.1 Managing Users And Groups With Base View

Within Base View:

- users can be managed via the navigation path `Identity Management` > `Users`
- groups can be managed via the navigation path `Identity Management` > `Groups`.

For users (figure 6.1) the LDAP entries for users are displayed. These entries are editable and each user can then be managed in further detail.

There is already one user on a newly-installed BCM: `cmsupport`. This user has no password set by default, which means (section 6.2.2) no logins to this account are allowed by default. BCM uses the user `cmsupport` to run various diagnostics utilities, so it should not be removed, and the default contents of its home directory should not be removed.

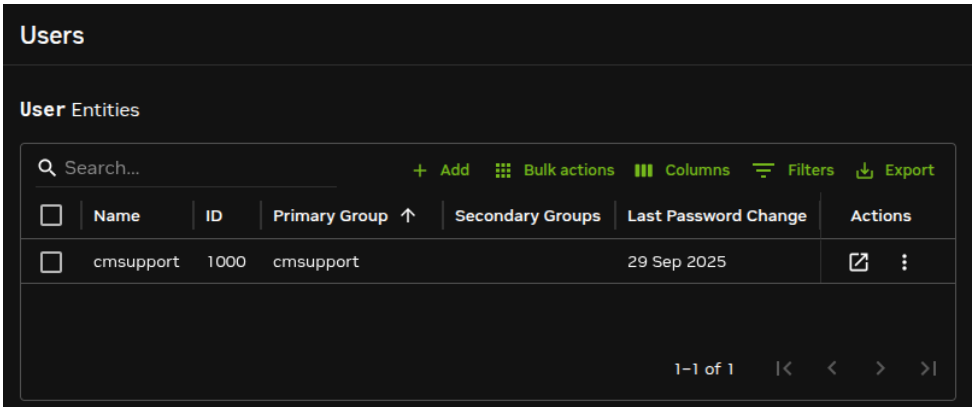


Figure 6.1: Base View User Management

The `+ Add` button allows users to be added via a subwindow (figure 6.2). Parameter values can be committed for the users via the `Commit User` button in the subwindow.

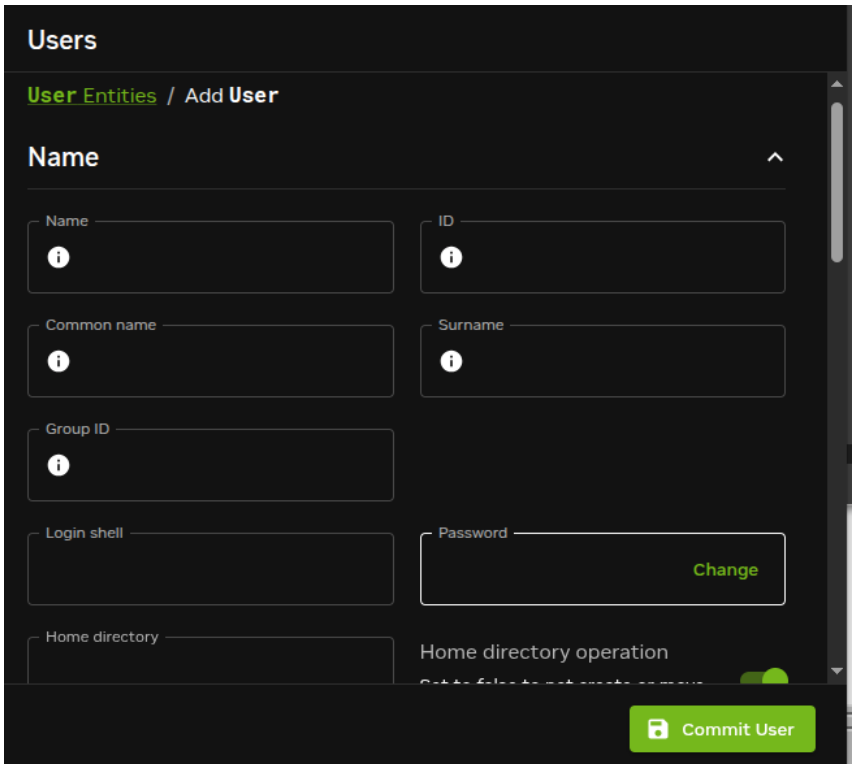


Figure 6.2: Base View User Management: Add Dialog

When saving an addition or modification:

- User and group ID numbers are automatically assigned from UID and GID 1000 onward (500 onward on SLES).
- A home directory is created and a login shell is set. Users with unset passwords cannot log in.

Group management in Base View is carried out via the navigation path `Identity Management` > `Groups`. Clickable LDAP object entries for regular groups then show up, similar to the user entries already covered. Management of these entries is done with the same functions as for user management.

6.2 Managing Users And Groups With cmsh

User management tasks as carried out by Base View in section 6.1, can be carried with the same end results in cmsh too.

A cmsh session is run here in order to cover the functions corresponding to the user management functions of Base View of section 6.1. These functions are run from within the user mode of cmsh:

Example

```
[root@basecm11 ~]# cmsh
[basecm11]% user
[basecm11->user]%
```

6.2.1 Adding A User

This part of the session corresponds to the functionality of the Add button operation in section 6.1. In user mode, the process of adding a user maureen to the LDAP directory is started with the add command:

Example

```
[basecm11->user]% add maureen
[basecm11->user*[maureen*]]%
```

The cmsh utility helpfully drops into the user object just added, and the prompt shows the user name to reflect this. Going into user object would otherwise be done manually by typing use maureen at the user mode level.

Asterisks in the prompt are a helpful reminder of a modified state, with each asterisk indicating that there is an unsaved, modified property at that asterisk's level.

The modified command displays a list of modified objects that have not yet been committed:

Example

```
[basecm11->user*[maureen*]]% modified
State  Type                    Name
------ ----------------------- ---------------
+      User                    maureen
```

Running show at this point reveals a user name entry, but empty fields for the other properties of user maureen. So the account in preparation, while it is modified, is clearly not yet ready for use:

Example

```
[basecm11->user*[maureen*]]% show
Parameter                      Value
------------------------------ ------------------------------------------------
Accounts
Managees
Name                           maureen
Primary group
Revision
Secondary groups
ID
Common name
Surname
Group ID
Login shell
Password                       < not set >
Home directory
```

```
Home directory operation            yes
Email
Profile
Write ssh proxy config              no
Create ssh key                      no
Disable password ssh                no
Allow GPU workload power profiles   no
Authorized ssh keys                 <0B>
Shadow min                          0
Shadow max                          999999
Shadow warning                      7
Shadow inactive                     0
Last change                         1970/1/1
Expiration date                     2038/1/1
Project manager                     <submode>
Notes                               <0B>
```

6.2.2 Saving The Modified State

This part of the session corresponds to the functionality of the SAVE button operation in section 6.1.

In the preceding section 6.2.1, user maureen was added. maureen now exists as a proposed modification, but has not yet been committed to the LDAP database.

Running the commit command now at the maureen prompt stores the modified state at the user maureen object level:

Example

```
[basecm11->user*[maureen*]]% commit
[basecm11->user[maureen]]% show
Parameter                           Value
----------------------------------- ------------------------------------------------
Accounts
Managees
Name                                maureen
Primary group                       1001
Revision
Secondary groups
ID                                  1001
Common name                         maureen
Surname                             maureen
Group ID                            1001
Login shell                         /bin/bash
Password                            ********
Home directory                      /home/maureen
Home directory operation            yes
Email
Profile
Write ssh proxy config              no
Create ssh key                      no
Disable password ssh                no
Allow GPU workload power profiles   no
Authorized ssh keys                 <0B>
Shadow min                          0
Shadow max                          999999
Shadow warning                      7
Shadow inactive                     0
Last change                         2025/5/20
```

```
Expiration date                    2038/1/1
Project manager                    <submode>
Notes                              <0B>
```

If, however, commit were to be run at the user mode level without dropping into the maureen object level, then instead of just that modified user, all modified users would be committed.

When the commit is done, all the empty fields for the user are automatically filled in with defaults based the underlying Linux distribution used. Also, as a security precaution, if an empty field (that is, a "not set") password entry is committed, then a login to the account is not allowed. So, in the example, the account for user maureen exists at this stage, but still cannot be logged into until the password is set. Editing passwords and other properties is covered in section 6.2.3.

The default permissions for file and directories under the home directory of the user are defined by the umask settings in /etc/login.defs, as would be expected if the administrator were to use the standard useradd command. Setting a path for the homedirectory parameter for a user sets a default home directory path. By default the default path is /home/<*username*> for a user <*username*>. If homedirectory is unset, then the default is determined by the HomeRoot directive (Appendix C).

6.2.3 Editing Properties Of Users And Groups

This corresponds roughly to the functionality of the Edit operation in section 6.1.

In the preceding section 6.2.2, a user account maureen was made, with an unset password as one of its properties. Logins to accounts with an unset password are refused. The password therefore needs to be set if the account is to function.

Editing Users With set And clear

The tool used to set user and group properties is the set command. Typing set and then either using tab to see the possible completions, or following it up with the enter key, suggests several parameters that can be set, one of which is password:

Example

```
[basecm11->user[maureen]]% set
Name:
          set - Set specific user property

Usage:
          set [OPTIONS] [user] <parameter> <value> [<value> ...]

Options:
          -e, --extra
               Set an extra free key/value parameter

          -v, --vector
               Set extra parameter values as a vector even for a single value

          -t, --type <type>
               Convert extra parameter values, type: [i]nt, [u]nsigned, f[float], d[double]

Arguments:
          user                name of the user, omit if current is set

Parameters:
          name ................ User login (e.g. donald)
          id .................. User ID number
          commonname .......... Full name (e.g. Donald Duck)
          surname ............. Surname (e.g. Duck)
```

```
          groupid ............. Base group of this user
          loginshell .......... Login shell
          homedirectory ....... Home directory
          password ............ Password
          homedirectoryoperation  Set to false to not create or move home directory
          shadowmin ........... Minimum number of days required between password changes
          shadowmax ........... Maximum number of days for which the user password remains valid.
          shadowwarning ....... Number of days of advance warning given to the user before the user
                                password expires
          shadowinactive ...... Number of days of inactivity allowed for the user
          expirationdate ...... Date on which the user login will be disabled
          email ............... Email
          profile ............. Profile for Authorization
          projectmanager ...... Project manager
          notes ............... Administrator notes
          writesshproxyconfig . Write ssh proxy config
          createsshkey ........ Create ssh key for added users
          disablepasswordssh .. Disable password ssh
          authorizedsshkeys ... Authorized ssh keys
          allowgpuworkloadpowerprofiles  Allow changing GPU workload power profiles from jobs
          revision ............ Entity revision
[basecm11->user[maureen]]%
```

Continuing the session from the end of section 6.2.2, the password can be set at the user context prompt like this:

Example

```
[basecm11->user[maureen]]% set password seteca5tr0n0my
[basecm11->user*[maureen*]]% commit
[basecm11->user[maureen]]%
```

At this point, the account `maureen` is finally ready for use.

The converse of the `set` command is the `clear` command, which clears properties:

Example

```
[basecm11->user[maureen]]% clear password; commit
```

Setting A Password With The LDAP Hash

Setting a password in `cmsh` is also possible by setting the LDAP hash (the encrypted storage format) that is generated from the password within `cmsh`. When setting passwords in `cmsh`, a string starting with `{MD5}`, `{CRYPT}` or `{SSHA}` is considered to be the hash of the password:

Example

```
[root@basecm11 ~]#  #first create the LDAP salted SHA-1 hash of the password:
[root@basecm11 ~]# /cm/local/apps/openldap/sbin/slappasswd -h {SSHA} -s seteca5tr0n0my
[root@basecm11 ~]# {SSHA}sViD+lfSTtlIyOMuGwPGfGd5XKHgEm5d
[root@basecm11 ~]# cmsh
[basecm11]% user use maureen
[basecm11->user[maureen]]% set password
enter new password:          #here and in the next line {SSHA}sViD+lfSTtlIyOMuGwPGfGd5XKHgEm5d is typed in
retype new password:
[basecm11->user[maureen]]% commit
[basecm11->user[maureen]]% !ssh maureen@node001      #now will test the password that generated the hash
Warning: Permanently added 'node001' (ECDSA) to the list of known hosts.
```

```
maureen@node001's password:   #here seteca5tr0n0my is typed in
Creating ECDSA key for ssh
[maureen@node001 ~]$          #successfully logged in with the password associated with the hash
```

Managing passwords in cmsh via direct LDAP hash entry is not normally done.

Forcing The User To Change A Set Password On Login

The AdvancedConfig AddUserPasswordReset (page 930) can be used to force a user password change. This can be for new users on first login, or for existing users.

Setting The Starting UID

The AdvancedConfig LowUserId (page 938) controls the starting point for auto-allocated UIDs and GIDs.

Editing Groups With append And removefrom

While the preceding commands set and clear also work with groups, there are two other commands available which suit the special nature of groups. These supplementary commands are append and removefrom. They are used to add extra users to, and remove extra users from a group.

For example, it may be useful to have a printer group so that several users can share access to a printer. For the sake of this example (continuing the session from where it was left off in the preceding), tim and fred are now added to the LDAP directory, along with a group printer:

Example

```
[basecm11->user[maureen]]% add tim; add fred
[basecm11->user*[fred*]]% exit; group;  add printer
[basecm11->group*[printer*]]% commit
[basecm11->group[printer]]% exit; exit; user
[basecm11->user*]%
```

The context switch that takes place in the preceding session should be noted: The context of user maureen was eventually replaced by the context of group printer. As a result, the group printer is committed, but the users tim and fred are not yet committed, which is indicated by the asterisk at the user mode level.

Continuing onward, to add users to a group the append command is used. A list of users maureen, tim and fred can be added to the group printer like this:

Example

```
[basecm11->user*]% commit
Successfully committed 2 Users
[basecm11->user]% group use printer
[basecm11->group[printer]]% append members maureen tim fred; commit
[basecm11->group[printer]]% show
Parameter                Value
------------------------ --------------------------
ID                       1002
Revision
Name                     printer
Members                  maureen,tim,fred
```

To remove users from a group, the removefrom command is used. A list of specific users, for example, tim and fred, can be removed from a group like this:

```
[basecm11->group[printer]]% removefrom members tim fred; commit
[basecm11->group[printer]]% show
Parameter                Value
```

```
---------------------- -------------------------
ID                     1002
Revision
Name                   printer
Members                maureen
```

The `clear` command can also be used to clear members—but it also clears all of the extras from the group:

Example

```
[basecm11->group[printer]]% clear members
[basecm11->group*[printer*]]% show
Parameter              Value
---------------------- -------------------------
ID                     1002
Revision
Name                   printer
Members
```

The `commit` command is intentionally left out at this point in the session in order to illustrate how reversion is used in the next section.

6.2.4 Reverting To The Unmodified State

This corresponds roughly to the functionality of the `Revert` operation in section 6.1.

This section (6.2.4) continues on from the state of the session at the end of section 6.2.3. There, the state of group `printers` was cleared so that the extra added members were removed. This state (the state with no group members showing) was however not yet committed.

The `refresh` command (page 61) reverts an uncommitted object back to the last committed state.

This happens at the level of the object it is using. For example, the object that is being handled here is the properties of the group object `printer`. Running `revert` at a higher level prompt—say, in the `group` mode level—would revert everything at that level and below. So, in order to affect only the properties of the group object `printer`, the `refresh` command is used at the group object `printer` level prompt. It then reverts the properties of group object `printer` back to their last committed state, and does not affect other objects:

Example

```
[basecm11->group*[printer*]]% refresh
[basecm11->group[printer]]% show
Parameter              Value
---------------------- -------------------------
ID                     1002
Revision
Name                   printer
Members                maureen
```

Here, the user `maureen` reappears because she was stored in the last save. Also, because only the group object `printer` has been committed, the asterisk indicates the existence of other uncommitted, modified objects.

6.2.5 Removing A User

Removing a user using `cmsh` corresponds roughly to the functionality of the `Delete` operation in section 6.1.

The `remove` command removes a user or group. The useful "`-d|--data`" flag added to the end of the username removes the user's home directory too. For example, within `user` mode, the command

"`remove user maureen -d; commit`" removes user `maureen`, along with her home directory. Continuing the session at the end of section 6.2.4 from where it was left off, as follows, shows this result:

Example

```
[basecm11->group[printer]]% user use maureen
[basecm11->user[maureen]]% remove -d;  commit
Successfully removed 1 Users
Successfully committed 0 Users
[basecm11->user]% !ls -d /home/*| grep maureen   #no maureen left behind
[basecm11->user]%
```

6.3 Using An External LDAP Server

Sometimes, an external LDAP server is used to serve the user database. If, instead of just using the database for authentication, the user database is also to be managed, then its LDAP schema must match the BCM LDAP schema.

For RHEL8, the `/etc/nslcd.conf`, `/etc/openldap/ldap.conf`, and the certificate files under `/cm/local/apps/openldap/etc/certs/` should be copied over. For RHEL9, `/etc/sssd/sssd.conf` is copied over instead of `/etc/nslcd.conf`.

Port 636 on Shorewall running on the head node should be open for LDAP communication over the external network, if external nodes are using it on the external network. In addition, the external nodes and the head node must be able to resolve each other.

By default, BCM runs an LDAP health check using the `cmsupport` user on the LDAP server. The LDAP health check may need to be modified or disabled by the administrator to prevent spurious health warnings with an external LDAP server:

Modifying Or Disabling The `ldap` Healthcheck

Modifying the `ldap` health check: To keep a functional `ldap` health check with an external LDAP server, a permanent external LDAP user name, for example `ldapcheck`, can be added. This user can then be set as the parameter for BCM's `ldap` health check object that is used to monitor the LDAP service. Health checks and health check objects are discussed in Chapter 10.

- If user management is not configured to work on CMDaemon for the external LDAP server, then the user management tool that is used with the external LDAP server should be used by the administrator to create the `ldapcheck` user instead.
- If user management is still being done via CMDaemon, then an example session for configuring the `ldap` script object to work with the new external LDAP user is (some prompt text elided):

  Example

  ```
  [root@basecm11 ~]# cmsh
  [basecm11]% user
  [basecm11->user]% add ldapcheck; commit
  [basecm11->user[ldapcheck]]% monitoring setup use ldap
  [basecm11->monitoring->setup[ldap]]% show
  Parameter                      Value
  ------------------------------ ----------------------------------------
  ...
  Arguments
  ...
  [basecm11->monitoring->setup[ldap]]% set arguments "ldapcheck"; commit
  [basecm11->monitoring->setup[ldap:ldapcheck]]%
  ```

Disabling the `ldap` health check: Instead of modifying the `ldap` health check to work when using an external LDAP server, it can be disabled entirely via Base View or `cmsh`.

- Base View: the `ldap` health check is disabled via the navigation path:

  `Monitoring` > `Data Producers` > `ldap` > `Edit`

- `cmsh`: the `disabled` parameter of the `ldap` health check object is set to `yes`. The `disabled` parameter for the `ldap` health check can be set as follows:

```
[root@basecm11 ~]# cmsh -c "monitoring setup use ldap; set disabled yes; commit"
```

Configuring The Cluster To Authenticate Against An External LDAP Server

The cluster can be configured in different ways to authenticate against an external LDAP server.

For smaller clusters, a configuration where LDAP clients on all nodes point directly to the external server is recommended. An easy way to set this up is as follows:

- On the head node:
  - In distributions that are derived from the RHEL 8.*x* series: The files in which the changes need to be made are `/etc/nslcd.conf` and `/etc/openldap/ldap.conf`. To implement the changes, the `nslcd` daemon must then be restarted, for example with `systemctl nslcd restart`.

    For the RHEL 9.*x* series, `/etc/nslcd.conf` is replaced by `/etc/sssd/sssd.conf`, and it is the `sssd` daemon that must be restarted, for example with: `/bin/systemctl restart sssd`.
  - the `updateprovisioners` command (section 5.2.4) is run to update any other provisioners.
- Then, the configuration files are updated in the software images that the nodes use. If the nodes use the `default-image`, and if the nodes are based on RHEL8 and derivatives, then the files to update are `/cm/images/default-image/etc/nslcd.conf` and `/cm/images/default/etc/openldap/ldap.conf`.

  For RHEL9 and derivatives `/cm/images/default-image/etc/sssd/sssd.conf` is used instead `/cm/images/default-image/etc/nslcd.conf`.

  After the configuration change has been made, and the nodes have picked up the new configuration, the regular nodes can then carry out LDAP lookups.
  - Nodes can simply be rebooted to pick up the updated configuration, along with the new software image.
  - Alternatively, to avoid a reboot, the `imageupdate` command (section 5.6.2) can be run to pick up the new software image from a provisioner.
- The CMDaemon configuration file `cmd.conf` (Appendix C) has LDAP user management directives. These may need to be adjusted:
  - If another LDAP tool is to be used for external LDAP user management instead of Base View or `cmsh`, then altering `cmd.conf` is not required, and BCM's user management capabilities do nothing in any case.
  - If, however, system users and groups are to be managed via Base View or `cmsh`, then CMDaemon, too, must refer to the external LDAP server instead of the default LDAP server. This configuration change is actually rare, because the external LDAP database schema is usually an existing schema generated outside of BCM, and so it is very unlikely to match BCM LDAP database schema. To implement the changes:

* On the node that is to manage the database, which is normally the head node, the `LDAPHost`, `LDAPUser`, `LDAPPass`, and `LDAPSearchDN` directives in `cmd.conf` are changed so that they refer to the external LDAP server.
* CMDaemon is restarted to enable the new configurations.

For larger clusters the preceding solution can cause issues due to traffic, latency, security and connectivity fault tolerance. If such occur, a better solution is to replicate the external LDAP server onto the head node, hence keeping all cluster authentication local, and making the presence of the external LDAP server unnecessary except for updates. This optimization is described in the next section.

6.3.1 External LDAP Server Replication

This section explains how to set up replication for an external LDAP server to an LDAP server that is local to the cluster, if improved LDAP services are needed. Section 6.3.2 then explains how this can then be made to work with a high availability setup.

Typically, the BCM LDAP server is configured as a replica (consumer) to the external LDAP server (provider), with the consumer refreshing its local database at set timed intervals. How the configuration is done varies according to the LDAP server used. The description in this section assumes the provider and consumer both use OpenLDAP.

External LDAP Server Replication: Configuring The Provider

It is advisable to back up any configuration files before editing them.

The provider is assumed to be an external LDAP server, and not necessarily part of the BCM cluster. The LDAP TCP ports `389` and `689` may therefore need to be made accessible between the consumer and the provider by changing firewall settings.

If a provider LDAP server is already configured then the following synchronization directives must be in the `slapd.conf` file to allow replication:

```
index entryCSN eq
index entryUUID eq
overlay syncprov
syncprov-checkpoint <ops> <minutes>
syncprov-sessionlog <size>
```

The openldap documentation (`http://www.openldap.org/doc/`) has more on the meanings of these directives. If the values for <*ops*>, <*minutes*>, and <*size*> are not already set, typical values are:

```
syncprov-checkpoint 1000 60
```

and:

```
syncprov-sessionlog 100
```

To allow the consumer to read the provider database, the consumer's access rights need to be configured. In particular, the `userPassword` attribute must be accessible. LDAP servers are often configured to prevent unauthorized users reading the `userPassword` attribute.

Read access to all attributes is available to users with replication privileges. So one way to allow the consumer to read the provider database is to bind it to replication requests.

Sometimes a user for replication requests already exists on the provider, or the root account is used for consumer access. If not, a user for replication access must be configured.

A replication user, `syncuser` with password `secret` can be added to the provider LDAP with adequate rights using the following `syncuser.ldif` file:

```
dn: cn=syncuser,<suffix>
objectClass: person
cn: syncuser
sn: syncuser
userPassword: secret
```

Here, <*suffix*> is the suffix set in `slapd.conf`, which is originally something like `dc=example,dc=com`. The `syncuser` is added using:

```
ldapadd -x -D "cn=root,<suffix>" -W -f syncuser.ldif
```

This prompts for the root password configured in `slapd.conf`.

To verify `syncuser` is in the LDAP database the output of `ldapsearch` can be checked:

```
ldapsearch -x "(sn=syncuser)"
```

To allow access to the `userPassword` attribute for `syncuser` the following lines in `slapd.conf` are changed, from:

```
access to attrs=userPassword
  by self write
  by anonymous auth
  by * none
```

to:

```
access to attrs=userPassword
  by self write
  by dn="cn=syncuser,<suffix>" read
  by anonymous auth
  by * none
```

Provider configuration is now complete. The server can be restarted using

```
systemctl restart slapd.service
```

in RHEL8.*x* and RHEL9.*x*.

External LDAP Server Replication: Configuring The Consumer(s)

The consumer is an LDAP server on a BCM head node. It is configured to replicate with the provider by adding the following lines to `/cm/local/apps/openldap/etc/slapd.conf`:

```
syncrepl rid=2
  provider=ldap://external.ldap.server
  type=refreshOnly
  interval=01:00:00:00
  searchbase=<suffix>
  scope=sub
  schemachecking=off
  binddn="cn=syncuser,<suffix>"
  bindmethod=simple
  credentials=secret
```

Here:

- The `rid=2` value is chosen to avoid conflict with the `rid=1` setting used during high availability configuration (section 6.3.2).
- The provider argument points to the external LDAP server.
- The interval argument (format DD:HH:MM:SS) specifies the time interval before the consumer refreshes the database from the external LDAP. Here, the database is updated once a day.
- The credentials argument specifies the password chosen for the `syncuser` on the external LDAP server.

More on the `syncrepl` directive can be found in the openldap documentation (`http://www.openldap.org/doc/`).

The configuration files must also be edited so that:

- The <*suffix*> and `rootdn` settings in `slapd.conf` both use the correct <*suffix*> value, as used by the provider.
- The `base` value in `/etc/ldap.conf` uses the correct <*suffix*> value as used by the provider. This is set on all BCM nodes including the head node(s). If the `/etc/ldap.conf` file does not exist, then the note on page 334 applies.

Finally, before replication takes place, the consumer database is cleared. This can be done by removing all files, except for the `DB_CONFIG` file, from under the configured database directory, which by default is at `/var/lib/ldap/`.

The consumer is restarted using `systemctl restart slapd`. This replicates the provider's LDAP database, and continues to do so at the specified intervals.

6.3.2 High Availability

No External LDAP Server Case

If the LDAP server is not external—that is, if BCM is set to its high availability configuration, with its LDAP servers running internally, on its own head nodes—then by default LDAP services are provided from both the active and the passive node. The high-availability setting ensures that CMDaemon takes care of any changes needed in the `slapd.conf` file when a head node changes state from passive to active or vice versa, and also ensures that the active head node propagates its LDAP database changes to the passive node via a `syncprov/syncrepl` configuration in `slapd.conf`.

External LDAP Server With No Replication Locally Case

In the case of an external LDAP server being used, but with no local replication involved, no special high-availability configuration is required. The LDAP client configuration in `/etc/ldap.conf` simply remains the same for both active and passive head nodes, pointing to the external LDAP server. The file `/cm/images/default-image/etc/ldap.conf`, in each software image also point to the same external LDAP server. If the `/etc/ldap.conf` files referred to here in the head and software images do not exist, then the note on page 334 applies.

External LDAP Server With Replication Locally Case

In the case of an external LDAP server being used, with the external LDAP provider being replicated to the high-availability cluster, it is generally more efficient for the passive node to have its LDAP database propagated and updated only from the active node to the passive node, and not updated from the external LDAP server.

The configuration should therefore be:

- an active head node that updates its consumer LDAP database from the external provider LDAP server
- a passive head node that updates its LDAP database from the active head node's LDAP database

Although the final configuration is the same, the sequence in which LDAP replication configuration and high availability configuration are done has implications on what configuration files need to be adjusted.

1. For LDAP replication configuration done after high availability configuration, adjusting the new suffix in `/cm/local/apps/openldap/etc/slapd.conf` and in `/etc/ldap.conf` on the passive node to the local cluster suffix suffices as a configuration. If the `ldap.conf` file does not exist, then the note on page 334 applies.

2. For high availability configuration done after LDAP replication configuration, the initial LDAP configurations and database are propagated to the passive node. To set replication to the passive node from the active node, and not to the passive node from an external server, the provider option in the `syncrepl` directive on the passive node must be changed to point to the active node, and the suffix in `/cm/local/apps/openldap/etc/slapd.conf` on the passive node must be set identical to the head node.

The high availability replication event occurs once only for configuration and database files in BCM's high availability system. Configuration changes made on the passive node after the event are therefore persistent.

6.4 Keycloak Integration

Keycloak (keycloak.org) is an open-source identity and access manager. BCM can install and configure it on the head node and federate BCM's internal LDAP into a Keycloak realm, allowing other components to use Keycloak for single sign-on against BCM's existing user database. The scripts that carry this out are provided by the `cluster-tools` package, available from BCM 11.34 onwards.

6.4.1 Prerequisites

- A BCM 11.34 or later head node, with the setup carried out as `root`.
- Ubuntu 24.04 or Ubuntu 26.04 on the head node. The scripts use `apt` to install dependencies, and are not supported on RHEL or Rocky.
- MySQL or MariaDB already running, which is the case on a standard BCM head node.
- BCM's internal LDAP already running.
- Free ports `9443` (HTTPS) and `9999` (HTTP) on the head node.
- Outbound HTTPS access to GitHub, unless a Keycloak version is pre-staged locally.

Outbound HTTPS access to GitHub is used by `cm-setup-keycloak.py` to download and resolve the Keycloak version to install. This can be avoided by pre-staging a Keycloak release archive on the head node before running the script: `keycloak-`*`version`*`.tar.gz` should be placed under the directory given by `--install-dir` (which defaults to `/opt`), for example `/opt/keycloak-26.5.5.tar.gz`. The `--version` option must then be passed and must match the staged file's version, or the script still calls the GitHub API to resolve the latest version, which is the outbound access this prerequisite refers to:

Example

```
[root@basecm11 ~]# ls /opt/keycloak-26.5.5.tar.gz
/opt/keycloak-26.5.5.tar.gz
[root@basecm11 ~]# cm-setup-keycloak.py --admin-password <Keycloak (web console) admin password> \
    --db-password <MySQL database password for Keycloak> --version 26.5.5
```

The `--db-password` option sets the password for a MySQL user (named `keycloak` by default) that the script creates on the head node's MySQL server. Keycloak's own `keycloak.conf` then uses this password to connect to its `keycloak` database.

If `/opt/keycloak` (or the directory given by `--install-dir`) already exists from a previous installation, the download is skipped entirely regardless of `--version`. The `--bcm` option is a separate way to avoid the GitHub download: it fetches the Keycloak release from the internal BCM mirror instead of a locally pre-staged archive.

6.4.2 Installing Keycloak And Federating LDAP

The setup is carried out in two steps: installing Keycloak itself, then creating a realm that federates BCM's LDAP into it.

Example

```
[root@basecm11 ~]# module load cluster-tools
[root@basecm11 ~]# cm-setup-keycloak.py
[root@basecm11 ~]# cm-setup-keycloak-realm.py --ldap
```

`cm-setup-keycloak.py` installs OpenJDK, creates a MySQL database and user for Keycloak, downloads and builds Keycloak into `/opt/keycloak`, generates a self-signed TLS certificate, and creates and starts the `keycloak` systemd service. The default administrator credentials are `admin/system`, and the default database credentials are `keycloak/system`; both passwords should be set explicitly with the `--admin-password` and `--db-password` options rather than left at their defaults.

`cm-setup-keycloak-realm.py --ldap` then creates a realm, named after the head node's FQDN by default, creates the `linux-ssh` OpenID client, federates BCM's LDAP into that realm, sets up LDAP mTLS, and runs an initial full LDAP sync. The `--ldap` option is what selects this BCM-LDAP federation path; without it, the script instead sets up Keycloak-backed SSH authentication, which is a different use case.

6.4.3 Verifying The Installation

Example

```
[root@basecm11 ~]# systemctl status keycloak
[root@basecm11 ~]# cm-setup-keycloak.py --ready
[root@basecm11 ~]# cm-setup-keycloak-realm.py --ready
[root@basecm11 ~]# cm-keycloak-user.py --list
```

The `--ready` options exit `0` once Keycloak, and then the realm, are up. `cm-keycloak-user.py --list` should show the BCM LDAP users that were federated in. After adding a user in `cmsh`, running `cm-keycloak-ldap-sync.py --full` makes that user appear in the list; the first query after a sync can take up to around 30 seconds.

6.4.4 Related Commands

- `cm-keycloak-user.py --list`, `--add`, `--remove`, or `--modify`: manage realm users.
- `cm-keycloak-ldap-sync.py`: sync changed users; `--full` for a full sync.
- `cm-setup-keycloak-realm.py --users` or `--realms`: query realm state.

6.4.5 NVIDIA Mission Control Integration

NVIDIA Mission Control can federate its own LDAP into the Keycloak instance that the preceding steps install; this requires no additional BCM-side configuration. To connect NVIDIA Mission Control to it, provide the Keycloak URL (`https://<head-fqdn>:9443`), the realm name, the administrator credentials, and the Keycloak TLS certificate. If NVIDIA Mission Control is not running on the head node, port `9443/tcp` must be open to it in the head node firewall. The NVIDIA Mission Control side federation procedure is documented in the *NVIDIA Mission Control Manual*.

6.5 Tokens And Profiles

Access to Base View and `cmsh` is based on user certificates (section 2.3.3).

Tokens can be assigned by the administrator to users so that users can carry out some of the operations that the administrator does with Base View or `cmsh`. Every cluster management operation requires that each user, including the administrator, has the relevant tokens in their *profile* for the operation.

The tokens for a user are grouped into a profile, and such a profile is typically given a name by the administrator according to the assigned capabilities.

For example:

- The profile might be called `readmonitoringonly` if it allows the user to read the monitoring data only.
- The profile may be called `powerhandler` if the user is only allowed to carry out power operations.
- The preconfigured profile `admin` can be assigned to a regular BCM user, to give that user administrator capablilities for BCM. While `root` has the `admin` profile by default, it can be wise to avoid using the `root` account directly in BCM.

Each profile thus consists of a set of tokens, typically relevant to the name of the profile, and is typically assigned to several users.

The profile is stored as part of the authentication certificate (section 2.3) which is generated for running authentication operations to the cluster manager for the certificate owner.

Profiles are handled with the `profiles` mode of `cmsh`, or from the Base View `Profiles` window, accessible via a navigation path of `Identity Management` > `Profiles`

The following preconfigured profiles are available from `cmsh`:

Profile name	Default Tasks Allowed	`nonuser` ?
`admin`	all tasks	no
`autonomous-hardware-recovery`	autonomous hardware recovery tasks	yes
`autonomous-job-recovery`	autonomous job recovery tasks	yes
`bootstrap`	bootstrap tasks	yes
`cmhealth`	health-related prejob tasks	yes
`cmpam`	BCM PAM tasks	yes
`litenode`	CMDaemon Lite (section 2.6.7) tasks	yes
`monitoringpush`	pushing raw monitoring data to CMDaemon via a JSON POST (page 91 of the *Developer Manual*)	yes
`mqtt`	MQTT tasks	yes
`node`	node-related tasks, for example by the node-installer	yes
`portal`	user portal viewing	no
`power`	device power	yes
`prs`	PRS tasks	yes
`readonly`	view-only	no

The last column in the preceding table indicates whether the preconfigured profile is a `nonuser` profile or not. A `cmsh` one-liner that indicates this is:

```
[root@basecm11 ~]# cmsh -c "profile; foreach * (get name; get nonuser)" | paste - -
```

- Most of the preconfigured profiles are `nonuser` profiles. Such a profile is used by cluster manager clients, and should never be modified by the cluster administrator.
- The preconfigured profiles that are not `nonuser` profiles are `admin`, `readonly`, and `portal`. These can be modified by the cluster administrator and used for human users.

The cluster manager services that use the available preconfigured profiles can be viewed in `cmsh` the `list` command in `profile` mode.

The tokens, and other properties of a particular profile can be seen within `profile` mode as follows:

Example

```
[basecm11->profile]% show readonly
Parameter    Value
------------ ------------------------------------------------------------------------------
Name         readonly
Non user     no
Revision
Services     CMDevice CMNet CMPart CMMon CMJob CMAuth CMServ CMUser CMSession CMMain CMGui CMP+
Tokens       GET_DEVICE_TOKEN GET_CATEGORY_TOKEN GET_NODEGROUP_TOKEN POWER_STATUS_TOKEN GET_DE+
```

For screens that are not wide enough to view the parameter values, the values can also be listed:

Example

```
[basecm11->profile]% get readonly tokens
GET_DEVICE_TOKEN
GET_CATEGORY_TOKEN
GET_NODEGROUP_TOKEN
...
```

A profile can be set with `cmsh` for a user within `user` mode as follows:

Example

```
[root@basecm11 ~]# cmsh
[basecm11]% user use conner
[basecm11->user[conner]]% get profile

[basecm11->user[conner]]% set profile readonly; commit
```

Only a subset of the predefined profiles are available to users. The ones that are made available to users are `readonly`, `admin`, and `portal`.

6.5.1 Modifying Profiles

A profile can be modified by adding or removing appropriate tokens to it. For example, the `readonly` group by default has access to the burn status and burn log results. Removing the appropriate tokens stops users in that group from seeing these results.

In `cmsh` the removal can be done from within `profile` mode as follows:

```
[root@basecm11 ~]# cmsh
[basecm11]% profile use readonly
[...[readonly]]% removefrom tokens burn_status_token get_burn_log_token
[basecm11->profile*[readonly*]]% commit
```

Tab-completion after typing in `removefrom tokens` helps in filling in the tokens that can be removed.

In Base View (figure 6.3), the same removal action can be carried out via the navigation path:

`Identity Management > Profiles > Filters > readonly >` 

In the resulting display it is convenient to maximize the window. Also convenient is running a search for `burn`, which will show the relevant tokens:

`BURN_STATUS_TOKEN` and `GET_BURN_LOG_TOKEN`

as well as the subgroup they are in, which is the `CMDevice` subgroup.

The checkmarks can be removed from the `BURN_STATUS_TOKEN` and `GET_BURN_LOG_TOKEN` tags, and the changed profile can then be committed.

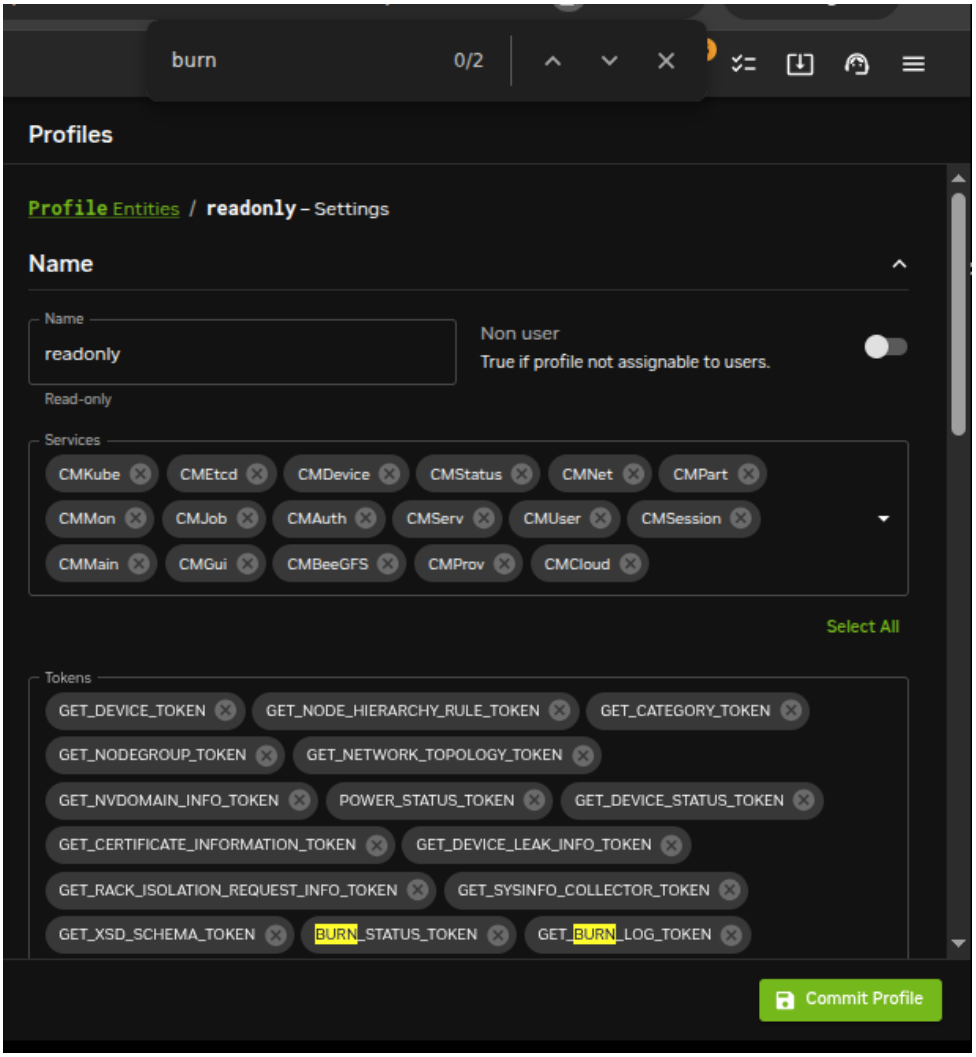


Figure 6.3: Base View Profile Token Management

6.5.2 Creation Of Custom Certificates With Profiles, For Users Managed By BCM's Internal LDAP

Custom profiles can be created to include a custom collection of capabilities in `cmsh` and Base View. Cloning of profiles is also possible from `cmsh`.

A certificate file, with an associated expiry date, can be created based on a profile. The time of expiry for a certificate cannot be extended after creation. An entirely new certificate is required after expiry of the old one.

The creation of custom certificates using `cmsh` (page 345) or Base View (page 346) is described later on. After creating such a certificate, the `openssl` utility can be used to examine its structure and properties. In the following example most of the output has been elided in order to highlight the expiry date (30 days from the time of generation), the common name (`democert`), the key size (2048), profile properties (`readonly`), and system login name (`peter`), for such a certificate:

```
[root@basecm11]# openssl x509 -in peterfile.pem -text -noout
    Data:
        ...
            Not After : Sep 21 13:18:27 2014 GMT
        Subject: ... CN=democert
                Public-Key: (2048 bit)
        ...
```

```
        X509v3 extensions:
            1.3.6.1.4.4324.1:
                ..readonly
            1.3.6.1.4.4324.2:
                ..peter
[root@basecm11]#
```

However, using the `openssl` utility for managing certificates is rather inconvenient. BCM provides more convenient ways to do so, as described next.

Listing Certificates

All certificates that have been generated by the cluster are noted by CMDaemon.

Listing certificates with `cmsh`: Within the `cert` mode of `cmsh`, the `listcertificates` command lists all cluster certificates and their properties:

```
[root@basecm11 ~]# cmsh
[basecm11]% cert
[basecm11-> cert]% listcertificates
Serial Revoked  Time left    Profile          System log in    Name
------ -------- ------------ ---------------- ---------------- ------------------------------
1      No       5214w 1d     admin            root             Administrator
2      No       5214w 1d     cmhealth                          CMHealth
3      No       5214w 1d     cmhealth                          CMHealth
4      No       5214w 1d     power                             Slurm
5      No       5214w 1d     bootstrap                         CertificateRequest
6      No       5214w 1d     cmpam                             CMPam
7      No       5214w 1d     portal                            WebPortal
...
```

Listing certificates with Base View: The Base View equivalent for listing certificates is via the navigation path `Identity Management` > `Certificates` (figure 6.4):

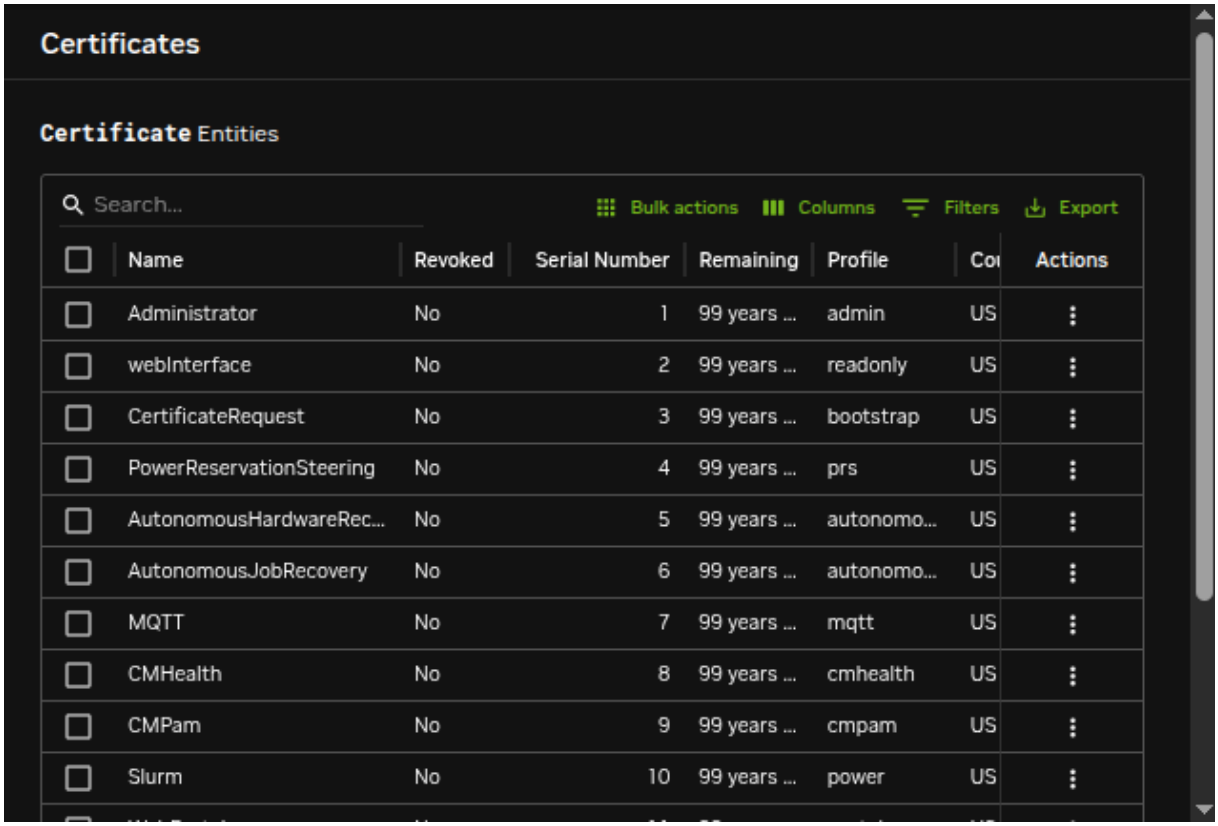


Figure 6.4: Base View Certificates List Window

Node Certificates

In the certificates list, node certificates that are generated by the node-installer (section 5.4.1) for each node for CMDaemon use are listed. These are entries with a `node`, and look like:

```
[basecm11-> cert]% listcertificates
Serial Revoked  Time left    Profile          System log in    Name
------ -------- ------------ ---------------- ---------------- ------------------------------
...
10     No       5214w 1d     node                              fa-16-3e-74-24-dc
11     No       5214w 1d     node                              fa-16-3e-57-2c-8e
12     No       5214w 1d     node                              fa-16-3e-b6-c7-4a
13     No       5214w 1d     node                              fa-16-3e-bd-cd-05
14     No       5214w 1d     node                              fa-16-3e-0d-ab-ea
...
```

Creating A Custom Certificate

Custom certificates are also listed in the certificates list.

Unlike node certificates, which are normally system-generated, custom certificates are typically generated by a user with the appropriate tokens in their profile, such as `root` with the `admin` profile. Such a user can create a certificate containing a specified profile, as discussed in the next section, by using:

- `cmsh`: with the `createcertificate` operation from within `cert` mode
- Base View: via the navigation path `Identity Management > Users >`  `> Profile` to set the `Profile`.

Creating a new certificate for `cmsh` users: Creating a new certificate in `cmsh` is done from `cert` mode using the `createcertificate` command, which has the following help text:

```
[basecm11->cert]% help createcertificate
Name:
      createcertificate - Create a new certificate

Usage:
      createcertificate <key-length> <common-name> <organization> <organizational-unit> <loca\
lity> <state> <country> <profile> <sys-login> <days> <key-file> <cert-file>

Arguments:
      key-file
            Path to key file that will be generated

      cert-file
            Path to pem file that will be generated
```

Accordingly, as an example, a certificate file with a read-only profile set to expire in 30 days, to be run with the privileges of user `peter`, can be created with:

Example

```
[basecm11->cert]% createcertificate 2048 democert a b c d ef readonly peter 30 /home/peter\
/peterfile.key /home/peter/peterfile.pem

Thu Jan  5 15:13:01 2023 [notice] basecm11: New certificate request with ID: 16
[basecm11->cert]% createcertificate 2048 democert a b c d ef readonly peter 30 /home/peter\
/peterfile.key /home/peter/peterfile.pem
Certificate key written to file: /home/peter/peterfile.key
Certificate pem written to file: /home/peter/peterfile.pem
```

The certificate list would show it as something like:

```
[basecm11-> cert]% listcertificates
Serial Revoked  Time left    Profile          System log in    Name
------ -------- ------------ ---------------- ---------------- ------------------------------
...
23     No       4w 1d        readonly         peter            democert
```

Setting the ownership of the new custom certificate: The certificates are owned by the owner generating them, so they are root-owned if `root` was running `cmsh`. This means that user `peter` cannot use them until their ownership is changed to user `peter`:

Example

```
[root@basecm11 ~]# cd /home/peter
[root@basecm11 peter]# ls -l peterfile.*
-rw------- 1 root root 1704 Aug 22 06:18 peterfile.key
-rw------- 1 root root 1107 Aug 22 06:18 peterfile.pem
[root@basecm11 peter]# chown peter:peter peterfile.*
```

Other users must have the certificate ownership changed to their own user names.

Associating users with paths to the new custom certificate: Users associated with such a certificate can then carry out `cmdaemon` tasks that have a read-only profile, and CMDaemon sees such users as being user `peter`. Two ways of being associated with the certificate are:

1. The paths to the `pem` and `key` files can be set with the `-i` and `-k` options respectively of `cmsh`. For example, in the home directory of `peter`, for the files generated in the preceding session, `cmsh` can be launched with these keys with:

```
[peter@basecm11 ~] cmsh -i peterfile.pem -k peterfile.key
[basecm11]% quit
```

2. If the `-i` and `-k` options are not used, then `cmsh` searches for default keys. The default keys for `cmsh` are under these paths under `$HOME`, in the following order of priority:

   (a) `.cm/admin.{pem,key}`

   (b) `.cm/cert.{pem,key}`

Creating a custom certificate for Base View users: As in the case of `cmsh`, a Base View user having a sufficiently privileged tokens profile, such as the `admin` profile, can create a certificate and key file for themselves or another user. This is done by associating a value for the `Profile` parameter for the user (figure 6.2).

The certificate files, `cert.pem` and `cert.key`, are then automatically placed in the following paths and names, under `$HOME` for the user:

- `.cm/admin.{pem,key}`
- `.cm/cert.{pem,key}`

Users that authenticate with their user name and password when running Base View use this certificate for their Base View clients, and are then restricted to the set of tasks allowed by their associated profile.

6.5.3 Creation Of Custom Certificates With Profiles, For Users Managed By An External LDAP

The use of an external LDAP server instead of BCM's for user management is described in section 6.3. Generating a certificate for an external LDAP user must be done explicitly in BCM. This can be carried out with the `external-user-cert` script, which is provided with the `cluster-tools` package. The package is installed by default with BCM.

Running the `external-user-cert` script embeds the user and profile in the certificate during certificate generation. The script has the following usage:

```
 external-user-cert -h
Usage: for a single profile: external-user-cert <profile> <user> [<user> ... ]
                             --home=<home-prefix> [-g <group>] [-o]
       for several profiles: external-user-cert --home=<home-prefix>
                             --file=<inputfile> [-g <group>]
                             where lines of <inputfile> have the syntax
                             <profile> <user> [<user> ... ]

Options:
  -h, --help        show this help message and exit
  --file=FILE       input FILE
  --home=HOME_PATH  path for home directories, default /home/
  -g GROUP          name of primary group, e.g. wheel
  -o                overwrite existing certificates
```

Here,

- <profile> should be a valid profile
- <user> should be an existing user
- <home-prefix> is usually /home
- <group> is a group, such as wheel
- <inputfile> is a file with each line having the syntax

  <profile> <user> [<user> ...]

One or more external LDAP user certificates can be created by the script. The certificate files generated are cert.pem and cert.key. They are stored in the home directory of the user.

For example, a user spongebob that is managed on the external server, can have a read-only certificate generated with:

```
# external-user-cert readonly spongebob --home=/home
```

If the home directory of spongebob is /home/spongebob, then the key files that are generated are /home/spongebob/.cm/cert.key and /home/spongebob/.cm/cert.pem.

Assuming no other keys are used by cmsh, a cmsh session that runs as user spongebob with readonly privileges can now be launched:

```
$ module load cmsh
$ cmsh
```

If other keys do exist, then they may be used according to the logic explained in item 2 on page 346.

6.5.4 Logging The Actions Of CMDaemon Users

The following directives allow control over the logging of CMDaemon user actions.

- CMDaemonAudit: Enables logging
- CMDaemonAuditorFile: Sets log location
- DisableAuditorForProfiles: Disables logging for particular profiles

Details on these directives are given in Appendix C.

6.5.5 Creation Of Certificates For Nodes With cm-component-certificate

The cm-component-certificate utility can be used to generate or update SSL certificates for components of services. The cluster administrator is not expected to use this utility because the cluster manager manages the certificates without bothering the administrator about it during normal operations. If the utilty is to be used, then it should be used with caution, to avoid failure in the components that use these certificates.

One of the SSL client components for which this utility works is LDAP.

Options include setting a new CA and creating a new certificate or key for nodes.

Some examples of how it can be used are:

LDAP PEM And Key Creation For A Standalone Node

A standalone node (page 972) is a node that is not provisioned from the head node, but is configured to boot from its own drive. A standalone node <*mynode*> may have a PEM and key certificate created on the head node with:

Example

```
[root@basecm11 ~]# cm-component-certificate --generate=<mynode>
Certificate saved to ./ldap.{pem,key}
Done.
```

The certificates are saved to the working directory. They should be copied over manually to the location of the LDAP certificates on *mynode*. The `nslcd` daemon on `mynode` for RHEL8 (or the `sssd` daemon for RHEL9) should then be restarted.

LDAP PEM And Key Creation For A Regular Node

If a node `node001` that is provisioned has a lost or corrupted LDAP key or certificate, then replacements for these can be made with:

Example

```
[root@basecm11 ~]# cm-component-certificate -n node001
Sending request to recreate certificates for 1 node to cmd on basecm11
[(38654705666, 1)] 1 0 0
1 certificates were successfully recreated
Done.
```

The `ldap.{pem,key}` files are automatically placed on `node001`, by default at the location specified by the CMDaemon LDAPCertificate and LDAPPrivateKey directives (page 911).

The files `/cm/node-installer/certificates/<node001-mac>/ldap.{pem,key}` should be removed on the head node.

The `nslcd`, `sssd`, and LDAP daemons should be restarted on `node001`, or more simply it can be rebooted if it is not in use. The reboot replaces the `ldap.{pem,key}` files on the head node with the newly-generated ones.

LDAP CA Certificate Creation

If a new LDAP CA certificate is needed, then a replacement can be made with:

```
[root@basecm11 ~]# cm-component-certificate --ca
Sending request to recreate the CA to cmd on basecm11
Done.
```

The following steps must be done manually:

- If there is another head node, then the CA files, by default ca.pem and `ca.key` under `/cm/local/apps/openldap/etc/certs/`, should be copied over the other head node.
- The key/PEM certificate files on all the nodes should be recreated using:
  `cm-component-certificate --allnodes`
- The old component PEM/key files for each regular node, `ldap.pem` and `ldap.key`, under the `node-installer` directory of the head node(s), should be removed. These certificates are kept under a directory named for the MAC address of the regular node, and follow the pattern:
  `/cm/node-installer/certificates/`<*MAC address*>`/`<*LDAP.{pem,key}*>
- The node CA files, by default `ca.pem` and `ca.key` under `/cm/local/apps/openldap/etc/certs/` should be copied to the nodes:

Example

```
[root@basecm11 ~]# export ldapcertdir="/cm/local/apps/openldap/etc/certs/"
for i in {01..12}
  do
    scp $ldapcertdir/ca.pem node0$i:/$ldapcertdir
    scp $ldapcertdir/ca.key node0$i:/$ldapcertdir
  done
```

- The `nslcd`, `sssd` and LDAP daemons should be restarted on all nodes
- CMDaemon should be restarted on the active head node

7

Workload Management

For clusters that have many users and a significant load, a workload manager (WLM) system allows a more efficient use of resources to be enforced for all users than if there were no such system in place. This is because without resource management, there is a tendency for each individual user to over-exploit common resources.

When a WLM is used, the end user can submit a job to it. This can be done interactively, but it is typically done as a non-interactive batch job.

The WLM assigns resources to the job, and checks the current availability as well as checking its estimates of the future availability of the cluster resources that the job is asking for. The WLM then schedules and executes the job based on the assignment criteria that the administrator has set for the WLM system. After the job has finished executing, the job output is delivered back to the user.

Among the hardware resources that can be used for a job are GPUs. Installing CUDA software to enable the use of GPUs is described in section 9 of the *Installation Manual*. Configuring GPU settings for BCM is described in section 3.17.2 of the *Administration Manual*. Configuring GPU settings for an individual WLM is described in the section on getting that particular WLM up and running.

The details of job submission from a user's perspective are covered in the *User Manual*.

Sections 7.1–7.5 cover the installation procedure to get a WLM up and running.

Sections 7.6 –7.7 describe how Base View and `cmsh` are used to view and handle jobs, queues and node drainage.

Section 7.8 shows examples of WLM assignments handled by BCM.

Section 7.9 describes the power saving features of WLMs.

Section 7.10 describes cgroups, a resources limiter, mostly in the context of WLMs.

Section 7.11 describes WLM customizations for settings other than the common settings covered by BCM.

7.1 Workload Managers Choices

Some WLM packages are installed by default, others require registration from the distributor before installation.

During cluster installation, a WLM can be chosen (figure 3.9 of the *Installation Manual*) for setting up. The choices are:

- **Slurm**: Available as version 25.05 or 25.11. Slurm is a free (GPL) job scheduler, with commercial support.
- **LSF v10.1**: IBM Spectrum LSF (Load Sharing Facility) version 10.1, is a further development of what used to be IBM Platform LSF.
- **None**: For clusters that need no HPC job-scheduling.

The WLMs in the preceding list can also be chosen and set up later using the cm-wlm-setup tool (section 7.3).

After installation, if there are no major changes in the WLM for updated versions of the workload managers, then

- WLMs that are packaged with BCM (Slurm) can have their packages updated using standard package update commands (yum update and similar). The installation and configuration of the WLM from the updated packages is carried out as described later on in this chapter.
- WLMs such as LSF that are installed by picking up software from the vendor can be updated by following vendor guidelines.

7.2 Forcing Jobs To Run In A Workload Management System

Another preliminary step is to consider forcing users to run jobs only within the WLM system. Having jobs run via a WLM is normally a best practice.

For convenience, BCM defaults to allowing users to log in via ssh to a node, using the authorized keys files stored in each users directory in /home (section 2.3.2). This allows users to run their processes without restriction, that is, outside the WLM system. For clusters with a significant load this policy results in a sub-optimal use of resources, since such unplanned-for jobs disturb any already-running jobs.

Disallowing user logins to nodes, so that users have to run their jobs through the WLM system, means that jobs are then distributed to the nodes only according to the planning of the WLM. If planning is based on sensible assignment criteria, then resources use is optimized—which is the entire aim of a WLM in the first place.

7.2.1 Disallowing User Logins To Regular Nodes Via cmsh

The usernodelogin setting of cmsh restricts direct user logins from outside the WLM, and is thus one way of preventing the user from using node resources in an unaccountable manner. The usernodelogin setting is applicable to node categories only, rather than to individual nodes.

In cmsh the attribute of usernodelogin is set from within category mode:

Example

```
[root@basecm11 ~]# cmsh
[basecm11]% category use default
[basecm11->category[default]]% set usernodelogin onlywhenjob
[basecm11->category*[default*]]% commit
```

The attributes for usernodelogin are:

- always (the default): This allows all users to ssh directly into a node at any time.
- never: This allows no user other than root to directly ssh into the node.
- onlywhenjob: This allows the user to ssh directly into the node when a job is running on it. It typically also prevents other users from doing a direct ssh into the same node during the job run, since typically the WLM is set up so that only one job runs per node. However, an ssh session that is already running is not automatically terminated after the job is done.
  - Some cluster administrators may wish to allow some special user accounts to override the onlywhenjob setting, for example for diagnostic purposes. Before giving the details of how to override the setting, some background explanation is probably useful:

    The onlywhenjob setting works with the PAM system, and adds the following line to /etc/pam.d/sshd on the regular nodes:

```
account         required        pam_bright.so
```

Nodes with the `onlywhenjob` restriction can be configured to allow a particular set of users to access them, despite the restriction, by allowing them with the PAM system, as follows: Within the software image <*node image*> used by the node, that is under `/cm/images/`<*node image*>, the administrator can add the set of user accounts to the file `etc/security/pam_bright.d/pam_allow.conf`. This file is installed in the software image with a chroot installation (section 9.4) of the `cm-libpam` package.

Groups of users can be allowed using the file `etc/security/pam_bright.d/pam_allow_group.conf`.

Other adjustments to PAM configuration, such as the number of attempted logins and the associated wait time per login, can be carried by editing the `/etc/security/pam_bright.d/cm-check-alloc.conf` file.

The image can then be updated to allow the users, by running the `imageupdate` command in `cmsh` (section 5.6.2), or by clicking the `Update node` option in Base View (section 5.6.3).

7.2.2 Disallowing User Logins To Regular Nodes Via Base View

In Base View, user node login access is set via a category setting, for example for the `default` category via the navigation path in figure 7.1:

`Grouping > Categories[default] >` ⧉ `> Settings > User node login`

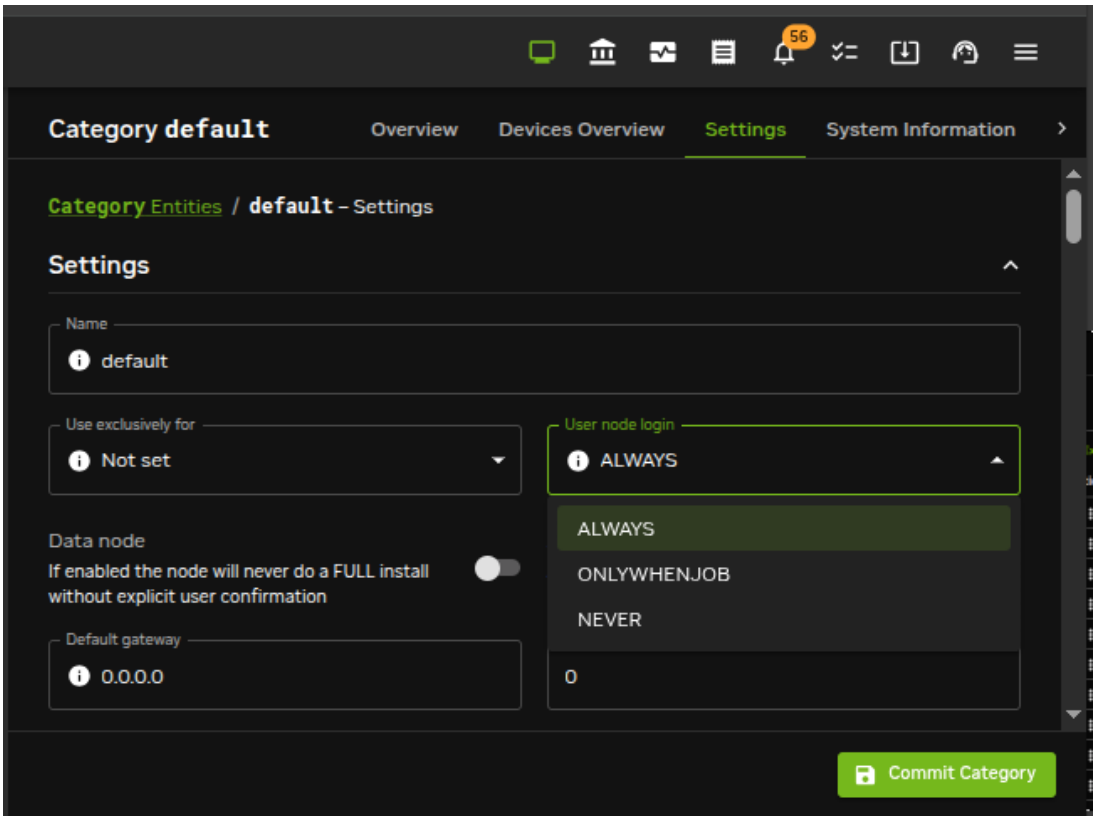


Figure 7.1: Disallowing user logins to nodes via Base View

7.2.3 Disallowing Other User Processes Outside Of Workload Manager User Processes

Besides disabling user logins, administrators may choose to disable interactive jobs in the WLM as an additional measure to prevent users from starting jobs on other nodes.

Administrators may also choose to set up scripts that run after job execution. Such scripts can terminate user processes outside the WLM, as part of a policy, or for general administrative hygiene. These are `Epilog` scripts and are part of the WLM.

The WLM documentation has more on configuring these options.

7.2.4 High Availability By Workload Managers

NVIDIA Base Command Manager uses the existing built-in high availability (HA) functionalities of workload managers as much as possible. A double server HA configuration with the existing built-in HA functionality can be carried out using the `cm-wlm-setup` utility (section 7.3), or using the Base View HA wizard (section 7.4.1). The built-in functionality makes use of the primary WLM server and secondary WLM server, which are placed on separate nodes.

The HA configuration aspect (section 15.1.3) in this case means that a WLM server role is assigned to both the WLM primary and the WLM secondary nodes. These primary WLM and secondary WLM servers can then:

- both be on head nodes,
- both be on regular (compute) nodes

It is not possible to configure a mixed setup—that is with one head node and one regular node—for the WLM servers on an HA setup.

In a non-HA cluster, the WLM `primaryserver` value is set to the head node by default.

Example

```
[head->wlm[slurm]]% get primaryserver
head
```

This is because NVIDIA Base Command Manager always configures the primary WLM server on the primary cluster head node.

If both the head nodes are configured with the WLM server roles, then the WLM `primaryserver` value is set to the primary cluster head node. This is because NVIDIA Base Command Manager always configures the primary WLM server on the primary cluster head node, and the backup WLM server on the secondary head node.

Example

```
[head1->wlm[slurm]]% get primaryserver
head1
[head1->wlm[slurm]]% exit
root@head1:~# ssh head2 "cmha makeactive"
...
...[logging into the newly active head node, and checking where the primary server is:]...
[head2->wlm[slurm]]% get primaryserver
head1
```

If the parameter `primaryserver` is unset in `wlm` mode, and the Slurm server role is assigned to the head nodes, then the `slurmctld` service is always started on both head nodes. During a failover, the Slurm configuration is regenerated and both `slurmctld` services are restarted. The primary `slurmctld` is always on the active head node.

If two regular nodes are configured with the WLM server roles instead, then in the cluster entity configuration the `primaryserver` parameter is set to one of the compute nodes.

The cluster primary head node and the WLM primary server should not be confused. In particular, the active cluster head node and active workload manager server are not necessarily the same for the case of the Slurm or LSF workload managers. For these, if the passive cluster head node becomes an

active cluster head node without a crash, then this does not trigger a passive WLM server on the newly active head node to also become an active WLM server. Thus, the WLM server can be active on a passive head node, and vice versa, because the WLM primary server is independent of the cluster primary head node.

Slurm High Availability, `scontrol takeover` And `slurmstartpolicy`

This section considers some Slurm high availability issues on running `scontrol takeover` with BCM integration.

Logical steps during the Slurm takeover between head nodes: Slurm high availability normally waits a short while with moving its control activity over to a passive head node that has just been made active. After a short delay, the Slurm failover algorithm is followed so that the Slurm controller (`slurmctld`) takes over operations on the newly active head node.

Slurm can speed up takeover by running `scontrol takeover` from the Bash shell on the node with the secondary `slurmctld`. Running `scontrol takeover` stops the primary `slurmctld` and triggers the Slurm failover algorithm.

This can be useful if the administrator wants to carry out maintenance on the usually active head node. Carrying out the takeover lets the Slurm service remain available, and stops the Slurm primary controller following the newly active head node.

The logic on how this works in BCM is:

1. The production state starts with
   - an active primary
   - a passive secondary
   - the primary head node set up as the primary Slurm controller
2. the administrator runs `cmha makeactive` on the secondary passive. This causes the following to happen:
   - `scontrol takeover` is run by CMDaemon on the secondary passive
   - CMDaemon goes into the passive state on the primary node, and `slurmctld` on the primary node stops
   - CMDaemon goes into the active state on the secondary node, and `slurmctld` on the secondary node starts
3. there is no interruption in Slurm services, because with `scontrol takeover` CMDaemon has forced the backup `slurmctld` to take over
4. the administrator can now carry out maintenance on the primary (now passive) head node
5. once maintenance has been completed, the administrator can optionally reboot the primary (passive) head node, and then make it active again with `cmha makeactive` to get back to the original production state

The problem with the preceding logic is that by default BCM automatically restarts a stopped `slurmctld` that it notices. This means that the results of `scontrol takeover` in steps 2 and 3 are unknown if carried out manually. That is, it is not practical to run the `scontrol takeover` command directly from Bash because it is possible to unintentionally configure the primary and backup `slurmctld` with an incorrect status.

To deal with this problem cleanly, CMDaemon provides the `slurmctldstartpolicy` parameter, described next.

The slurmctldstartpolicy parameter and Slurm takeover behavior: The slurmctldstartpolicy parameter can take the following values to decide the slurmctld behavior during scontrol takeover:

- TAKEOVER: The default value on an HA cluster.
- ALWAYS: The default value on a non-HA cluster.
- ACTIVEONLY: Useful only if /cm/shared is not mounted on both head nodes at the same time. For example, as in DAS (Direct Attached Storage), which should never be used anyway for shared storage in a production system.

1. If within NVIDIA Base Command Manager, in the SlurmServerRole, the slurmctldstartpolicy parameter is set to TAKEOVER, then disabling the automatic restart of the slurmctld service is unnecessary:

```
[root@basecm11 ~]# cmsh
[basecm11-]% configurationoverlay
[basecm11-]>configurationoverlay]% use slurm-server
[basecm11->configurationoverlay[slurm-server]]% roles
[basecm11->configurationoverlay[slurm-server]->roles]% use slurmserver
[basecm11->configurationoverlay[slurm-server]->roles[slurmserver]]% set slurmctldstartpolicy TAKEOVER
[basecm11->configurationoverlay*[slurm-server*]->roles*[slurmserver*]]% commit
```

   The setting value can be cleared with:

```
[basecm11->configurationoverlay[slurm-server]->roles[slurmserver]]% set slurmctldstartpolicy ALWAYS
[basecm11->configurationoverlay*[slurm-server*]->roles*[slurmserver*]]% commit
```

2. If cm-wlm-setup installs Slurm on an HA cluster, then the slurmctldstartpolicy parameter can be set in one of the TUI screens during the session. Before HA is configured, the parameter has a default value of ALWAYS. After HA is configured it has a default value of TAKEOVER. The default value of TAKEOVER causes prefailoverscript in the failover submode (section 15.4.6) to take the value /cm/local/apps/cmd/scripts/slurm.takeover.sh

3. If cm-wlm-setup installs Slurm on a non-HA cluster, then the prefailoverscript parameter must be set within the failover submode of partition mode:

```
[basecm11->partition[base]->failover]% set prefailoverscript /cm/local/apps/cmd/scripts/slurm.takeover.sh
[basecm11->partition*[base*]->failover*]% commit
```

   This ensures that the active head node is taken over by Slurm during a change of the active head node.

The takeover is temporary. If the head node which has the Slurm primary slurmctld on it is restarted, or if the service itself is restarted manually, then the primary slurmctld takes back its role, as per Slurm documentation.

For a Slurm cluster with a single slurmctld, where the Slurm server role is assigned to a single compute node or the BCM cluster has only one head node, the parameter slurmctldstartpolicy should be set to ALWAYS, which is the default.

Table 7.2.4 shows what slurmctld does when the slurmctldstartpolicy parameter is set to TAKEOVER and when in SlurmWlmCluster the parameter primaryserver is set. The detailed behavior per head node depends on whether the head nodes are active or primary:

Node pair	Is slurmctldstartpolicy TAKEOVER?	Active node?	Slurm primary node?	slurmctld starts?
Head001	no	*	*	yes
Head002	no	*	*	yes
Head001	yes	yes	yes	yes
Head002	yes	no	no	yes
Head001	yes	yes	no	yes
Head002	yes	no	yes	no
Head001	yes	no	yes	no
Head002	yes	yes	no	yes
Head001	yes	no	no	yes
Head002	yes	yes	yes	yes

* any value of yes or no

Table 7.2.4 slurmctldstartpolicy parameter and slurmctld

Slurm controller high availability on compute nodes: If the Slurm server role is assigned to two compute nodes, the TAKEOVER policy is configured automatically only when both nodes belong to a failover group. In this configuration, slurmctld does not start on the primary node while the secondary node is active.

7.3 Installation Of Workload Managers

Normally the administrator selects a WLM to be used during BCM installation (figure 3.9 of the *Installation Manual*). A WLM may however also be added and configured after BCM has been installed, by using cm-wlm-setup, which is part of the cm-setup package, or by using the Base View WLM wizard.

With most other objects, BCM front ends—cmsh and Base View—can be used to create a new object from scratch, or can clone a new object from another existing object. However, the front ends cannot do this for a WLM object. An attempt to create or clone a new WLM object using cmsh or Base View is prohibited by the front ends, because there are many pitfalls possible in configuration.

A new WLM object, and WLM instance, can therefore only be installed via cm-wlm-setup, the Base View WLM wizard, or by selecting a WLM during the initial BCM installation.

7.3.1 Running cm-wlm-setup In CLI Mode

The recommended way to run the cm-wlm-setup utility is without options or arguments, in which case a TUI dialog starts up. A TUI session run with cm-wlm-setup is covered in section 7.3.2.

However, the cm-wlm-setup utility can alternatively be used in a non-GUI, command-line, mode, with options and arguments. The utility has the following usage:

```
[root@basecm11 ~]# cm-wlm-setup -h
usage: Workload manager setup cm-wlm-setup [-c <config_file>]
                                           [--setup | --disable]
                                           [--wlm <name>]
                                           [--server-nodes SERVER_NODES]
```

```
[--server-primary SERVER_PRIMARY]
[--server-overlay-name SERVER_OVERLAY_NAME]
[--server-overlay-priority SERVER_OVERLAY_PRIORITY]
[--client-categories CLIENT_CATEGORIES]
[--client-nodes CLIENT_NODES]
[--client-overlay-name CLIENT_OVERLAY_NAME]
[--client-overlay-priority CLIENT_OVERLAY_PRIORITY]
[--client-slots <slots>]
[--submit-categories SUBMIT_CATEGORIES]
[--submit-nodes SUBMIT_NODES]
[--submit-overlay-name SUBMIT_OVERLAY_NAME]
[--submit-overlay-priority SUBMIT_OVERLAY_PRIORITY]
[--wlm-cluster-name WLM_CLUSTER_NAME]
[--reboot] [--reset-cgroups]
[--yes-i-really-mean-it]
[--archives-location <path>]
[--license <license>] [--purge]
[--accounting-overlay-name ACCOUNTING_OVERLAY_NAME]
[--accounting-overlay-priority ACCOUNTING_OVERLAY_PRIORITY]
[--with-pyxis]
[--add-pyxis]
[--reinstall-pyxis]
[--remove-pyxis]
[--pyxis-data-directory <path>]
[--nvidia-gpus NVIDIA_GPUS] [-v]
[--no-distro-checks] [--json]
[--output-remote-execution-runner]
[--on-error-action {debug,remotedebug,undo,abort}]
[--skip-packages]
[--min-reboot-timeout <reboot_timeout_seconds>]
[--allow-running-from-secondary]
[--dev] [-h]
```

The help output from running `cm-wlm-setup -h` continues on beyond the preceding text output, and presents more options.

These options can be grouped as follows:

Optional Arguments

- `--setup`: Helps set up a server, enable roles, and create the default queues/partitions
- `--disable`: Disable WLM services
- `-h, --help`: Displays the help screen

Common Arguments

- `-c` <*YAML configuration file*>: Loads a runtime configuration for plugins, from a YAML configuration file.

Options For Installing Or Managing A WLM

- `--wlm` <*WLM name*>: Specifies which WLM is to be set up. Choices for <*WLM name*> are:
  - `slurm`
  - `lsf`
- `--wlm-cluster-name` <*WLM cluster name*>: Specifies the name for the new WLM cluster that is to be set up.

- `--reboot`: Reboot after install

Server Role Settings

- `--server-nodes` <*server nodes*>: Sets the server roles of the WLM to the value set for <*server nodes*>, which is a comma-separated list of nodes. Default value: `HEAD`, which is a reserved name for the head node.
- `--server-primary` <*primary server*>: Sets the hostname used for the primary server to <*primary server*>. Default name: `HEAD`.
- `--server-overlay-name` <*server overlay name*>: Sets the server role configuration overlay name to <*server overlay name*>. Default name: <*WLM name*>`-server`, where <*WLM name*> is the name specified in the `--wlm` option.
- `--server-overlay-priority` <*server overlay priority*>: Sets the server role configuration overlay priority to <*server overlay priority*>. Default value: `500`.

Client Role Settings

- `--client-categories` <*client categories*>: Sets the client roles of the WLM to the value set for <*client categories nodes*>, which is a comma-separated list of node categories. Default value: `default`.
- `--client-nodes` <*client nodes*>: Sets the client roles of the WLM to the value set for <*client nodes*>, which is a comma-separated list of nodes. No value set by default.
- `--client-overlay-name` <*client overlay name*>: Sets the client role configuration overlay name to <*client overlay name*>. Default name: <*WLM name*>`-client`, where <*WLM name*> is the name specified in the `--wlm` option.
- `--client-overlay-priority` <*client overlay priority*>: Sets the client role configuration overlay priority to <*client overlay priority*>. Default value: `500`.
- `--client-slots` <*slots*>: Sets the number of slots on the client to <*slots*>.

Submit Role Settings

- `--submit-categories` <*submit categories*>: Sets the submit roles of the WLM to the value set for <*submit categories nodes*>, which is a comma-separated list of node categories that are submit nodes. Default value: `default`.
- `--submit-nodes` <*submit nodes*>: Sets the submit roles of the WLM to the value set for <*submit nodes*>, which is a comma-separated list of submit nodes. No value set by default.
- `--submit-overlay-name` <*submit overlay name*>: Sets the submit role configuration overlay name to <*submit overlay name*>. Default name: <*WLM name*>`-submit`, where <*WLM name*> is the name specified in the `--wlm` option.
- `--submit-overlay-priority` <*submit overlay priority*>: Sets the submit role configuration overlay priority to <*submit overlay priority*>. Default value: `500`.

Disable Options

- `--reset-cgroups`: Disable joining cgroup controllers with systemd setting JoinControllers
- `--yes-i-really-mean-it`: Required for additional safety

Workload Manager Specific Options

- `--archives-location` <*path*>: Set the directory path for the archive files, only for LSF. This parameter is mandatory for LSF installation.
- `--license` <*path*>: Set the path to the LSF license.
- `--purge`: Remove the directories on disable, for LSF.

Slurm Accounting Role Settings

- `--accounting-overlay-name` <*accounting overlay name*>: Sets the accounting role configuration overlay
- `--accounting-overlay-priority` <*accounting overlay priority*>: Sets the accounting role configuration overlay priority to <*accounting overlay priority*>. Default value: `500`.

Slurm Pyxis Settings

- `--add-pyxis`: Add Pyxis to an existing cluster
- `--reinstall-pyxis`: Reinstall or upgrade Pyxis on all the clusters using it
- `--remove-pyxis`: Remove Pyxis from an existing cluster
- `--with-pyxis`: Enable Pyxis
- `--pyxis-data-directory` <*path*>: Sets directory where images will be stored

Slurm GPU Settings

- `--nvidia-gpus` <*NVIDIA GPUs specification*>: Sets the NVIDIA GPUs that will be used in the configuration, specifying type and number of GPUs. For example: `--nvidia-gpus=A100:4`
- `--gpu-client-nodes` <*NVIDIA GPU client nodes*>: Sets a comma-separated list of nodes assigned to the GPU overlay.
- `--gpu-client-categories` <*NVIDIA GPU client categories*>: Sets a comma-separated list of node categories assigned to the GPU overlay.

Advanced Options

- `-v, --verbose`: This displays a more verbose output. It can be helpful in troubleshooting.
- `--no-distro-checks`: Disables distribution checks based on ds.json.
- `--json`: Use json formatting for logs printed to STDOUT.
- `--output-remote-execution-runner`: Format output for CMDaemon.
- `--on-error-action {debug,remotedebug,undo,abort}`: Upon encountering a critical error, instead of asking the user for choice, the setup will do the selected action.
- `--skip-packages`: Skip the stages which install packages. Requires packages to be already installed.
- `--min-reboot-timeout` <*timeout*>: How long to wait for nodes to finish reboot, in seconds. Minimum value: `300`. Default value: `300`.
- `--allow-running-from-secondary`: Allow the wizard to be run from the secondary when it is the active head node.
- `--dev`: Enables additional command line arguments for developers.

7.3.2 Running `cm-wlm-setup` As A TUI

Running `cm-wlm-setup` with no options and with no arguments brings up a TUI screen (figure 7.2).

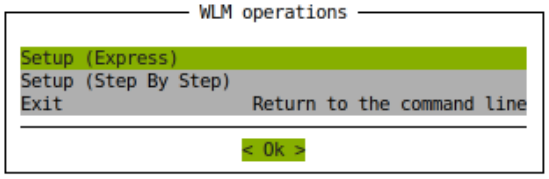


Figure 7.2: `cm-wlm-setup` TUI initial screen

Express Installation

The `Setup (Express)` menu option allows the administrator to select the workload manager in the next screen (figure 7.3), and to install it with a minimal number of configuration steps. If it has already been installed, but disabled using `cm-wlm-setup`, then it can also be re-enabled, instead of installed from scratch.

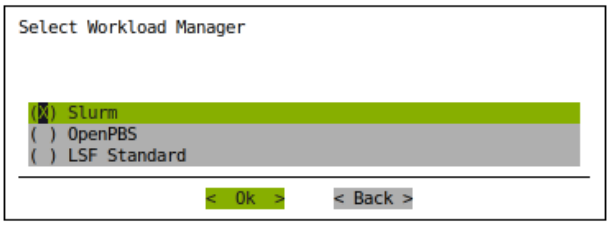


Figure 7.3: `cm-wlm-setup` TUI WLM selection screen

Step-by-step Installation

If the `Setup (Step By Step)` menu option is chosen instead of the express option, then this also allows the administrator to select the workload manager in figure 7.3. But after selection, there are a number of extra configuration steps that can be carried out which are not available in the express configuration. Guidance is given for these extra steps, and sensible default values are already filled in for many options.

One part of the step-by-step session involves assigning the WLM client role to the compute nodes of the cluster. Only the non-GPU compute nodes should be assigned the (standard, non-GPU) WLM client role. The GPU compute nodes are assigned a *GPU* WLM client role in the section of the TUI wizard that deals with GPU configuration.

The WLM client role is assigned to the entire category—the `default` category— of non-head nodes by default (figure 7.4):

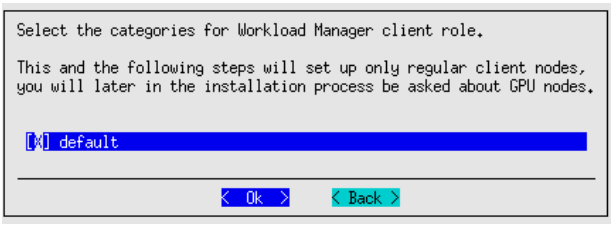


Figure 7.4: Slurm with `cm-wlm-setup`: WLM Client role category configuration screen

The WLM client role can be assigned to selected nodes only by setting the `default` category checkbox to blank, and then selecting the non-GPU nodes in the following screen. For example, as illustrated in figure 7.4, where the standard, non-GPU nodes are `node002` and `node003`:

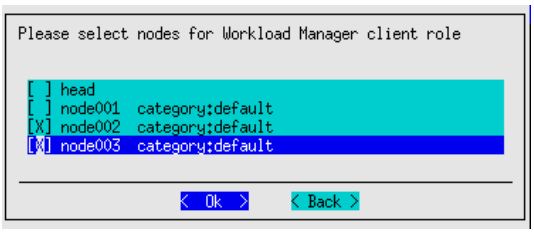


Figure 7.5: Slurm with `cm-wlm-setup`: WLM client role node configuration screen

GPU Configuration Screens

The GPU configuration screens are extra steps available during a `Setup (Step By Step)` session. The GPU configuration steps are discussed summarily in the quickstart section for GPUs, on page 13 of the

Installation Manual.

The GPU configuration steps are covered in this section in more detail:

After configuring the WLM server, WLM submission and WLM client roles for the nodes of the cluster, a screen that asks if GPU resources should be configured is displayed (figure 7.6):

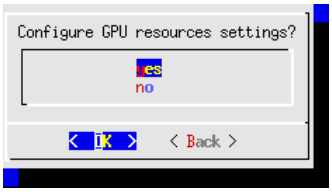


Figure 7.6: Slurm With `cm-wlm-setup`: GPU Configuration Entry Screen

Following through brings up a GPU device settings configuration screen (figure 7.7):

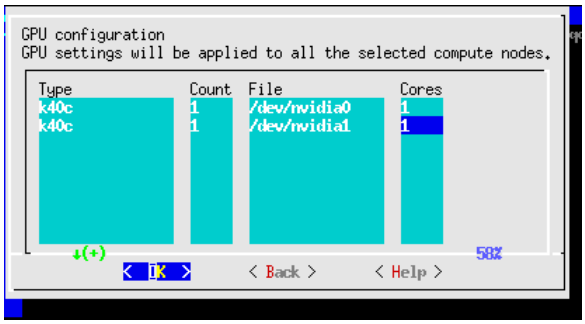


Figure 7.7: Slurm With `cm-wlm-setup`: GPU Device Settings Configuration Screen

The help text option in the screen gives hints based on the descriptions at `https://slurm.schedmd.com/gres.conf.html`, and also as seen in Slurm's `man gres.conf.5`.

Figure 7.7 shows 2 physical GPUs on the node being configured. The type is an arbitrary string for the GPU, and each CPU core is allocated an alias GPU device.

The next screen (figure 7.8) allows the NVIDIA CUDA MPS (Multi-Process Service) to be configured:

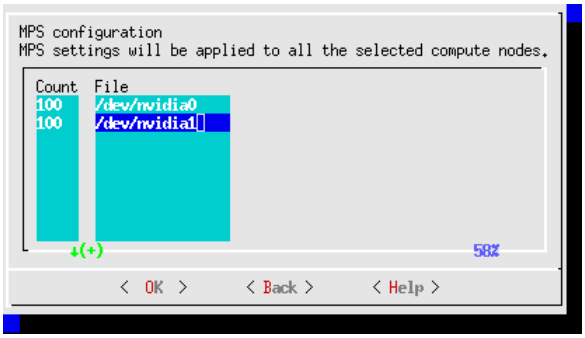


Figure 7.8: Configuring An NVIDIA GPU For Slurm With `cm-wlm-setup`: MPS Settings Configuration Screen

The help text for this screen gives hints on how the fields can be filled in. The number of GPU cores (figure 7.8) for a GPU device can be set.

Further screens allow:

- the configuration overlay (section 2.1.6) name to be set for the GPU WLM clients. By default the name is set to `slurm-client-gpu`.
- a GPU WLM client role to be assigned to a category, if, for example, all the GPU nodes have been

given their own category.

- the GPU WLM client role to be assigned to individual nodes instead of to a category.
- a configuration overlay priority to be set for a GPU WLM client role. By default, this has a value of 450.
- automatic GPU detection, with the following options:
  - BCM autodetects GPUs
  - Slurm autodetects NVIDIA GPU details
  - Slurm autodetects AMD GPU details
  - Slurm autodetects Intel GPU details
  - Slurm autodetects AWS Trainium/Inferentia devices
  - Manual GPU configuration in cm-wlm-setup
  - Skip GPU configuration

Slurm Accounting Database Configuration

In the step-by-step configuration for the Slurm WLM, after the server and client parameters have been set, the Slurm accounting database configuration can be set.

This requires setting:

- one or two accounting nodes
- a primary accounting node

All nodes must be reachable from slurmctld.

The accounting nodes can be set with the Select accounting nodes screen (figure 7.9):

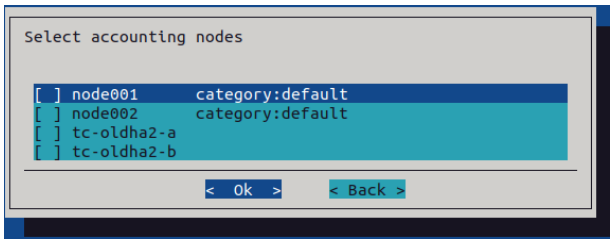


Figure 7.9: cm-wlm-setup selection of accounting nodes

Head nodes and compute nodes cannot be mixed for the accounting nodes. If two accounting nodes are set, then the slurmdbd high availability configuration is automatically set up.

The primary accounting node—that is, the node that is AccountingStorageHost in slurm.conf—can then be set in the Select the primary accounting server node screen (figure 7.10):

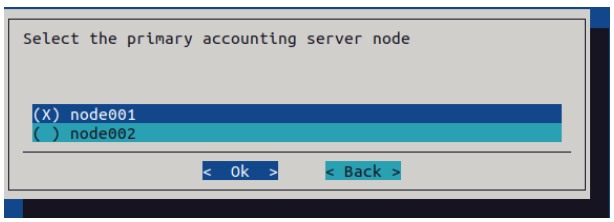


Figure 7.10: cm-wlm-setup selection of primary accounting node

The other accounting node is automatically set to be AccountingStorageBackupHost.

The type of node on which the slurmdbd database runs can then be set (figure 7.11):

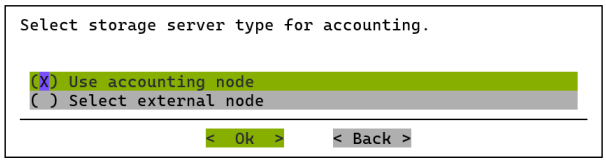


Figure 7.11: cm-wlm-setup selection of the type of node on which the Slurm accounting database runs

Only two of the three options are offered at a time. Which options are offered is determined by the accounting node selection made on the preceding screen. The accounting nodes are either all head nodes or all compute nodes. A mixed selection is rejected.

- If Use accounting node is selected, then the database is stored on the primary accounting node. This option is offered when the accounting nodes are head nodes
- If Select cluster node is selected, then the next screen allows the selection of the BCM MySQL overlay which is used to manage the database (figure 7.12). This option is offered when the accounting nodes are compute nodes, that is, when slurmdbd is offloaded from the head nodes. Whether the head nodes are configured for high availability makes no difference. If no MySQL configuration overlay is found, then the selection is rejected, and MySQL must first be installed with cm-mysql-setup. If the MySQL installation is running on more than one node (section 14.12), then mysql-router is automatically installed on all accounting nodes.

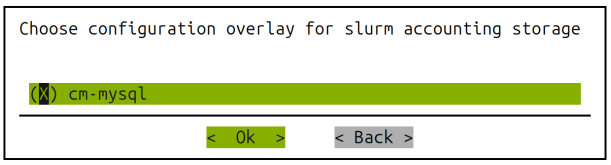


Figure 7.12: cm-wlm-setup Selection of host(s) for where the Slurm accounting database used by slurmdbd is stored

- If Select external node is selected, then a non-BCM hostname is asked for, and the cluster administrator is responsible for the installation and configuration of the DBMS. This option is always offered

Modifying the settings after installation is discussed on page 406.

Disabling An Installation

The Disable option in figure 7.2 allows the administrator to disable an existing instance.

Summary Screen

The screen that appears after the configuration steps are completed, is the Summary screen (figure 7.13). This screen allows the configuration to be viewed, saved, or saved and deployed.

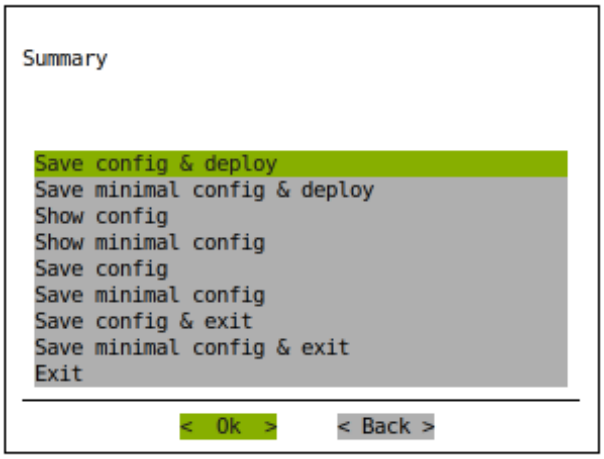


Figure 7.13: `cm-wlm-setup` TUI summary screen

In the final summary screen, as with other scripts that come with the `cm-setup` package, the configuration setting can be one of the following kinds:

- `minimal config`: For this minimal kind of configuration setting, the configuration consists only of the settings that differ from the default application configuration. The remaining default application configuration settings are not included in the configuration.
- `config`: For this non-minimal kind of configuration setting, the configuration consists of the minimal configuration settings (the ones changed by BCM), as well as the remaining default configuration settings (the ones provided with the application). That is to say: the remaining default configuration application settings are included in the configuration. The default application configuration settings may change as the upstream packages change.

If deployment is carried out, then several screens of output are displayed. After the deployment is completed, the log file can be viewed at `/var/log/cm-wlm-setup.log`.

7.3.3 Installation And Configuration Of Enroot And Pyxis With Slurm To Run Containerized Jobs

What Is Enroot?

As the README file for Enroot says, Enroot is an open source tool to turn container images into unprivileged sandboxes. Enroot can be thought of as an enhanced unprivileged chroot. It uses user and mount namespaces, as well as other modern kernel features, in order to create such sandboxes. It uses the same underlying technologies as containers, but removes much of the isolation that they inherently provide, while preserving filesystem separation. Further details on Enroot can be found at `https://github.com/NVIDIA/enroot`.

Enroot can be used with different workload managers, but for now only Slurm has been tightly integrated.

What Is Pyxis?

Pyxis (`https://github.com/NVIDIA/pyxis`) is a SPANK plugin for Slurm. SPANK (Slurm Plug-in architecture for Node and job (K)control, `man spank.8`) is a generic interface for job launch code control in Slurm. The Pyxis plugin requires the Enroot utility, and allows the user's jobs to be executed seamlessly over Enroot in unprivileged containers. The plugin enables the Slurm submission utilities to provide container-related command line options.

Enroot And Pyxis Packages

BCM provides two Enroot package flavors: *standard* and *hardened*. The standard binaries are compiled as follows:

- Open file descriptors are inherited
- Spectre variant 2 (IBPB/STIBP) mitigations are disabled
- Spectre variant 4 (SSBD) mitigations are disabled

As a rule of thumb: the hardened flavor is slightly more secure but suffers a larger overhead.

- The standard packages are already installed by default with BCM, and are:
  - `enroot`: provides the main utility and helper files.
  - `enroot+caps`: a nearly empty package which runs a post-installation script to grant extra capabilities to unprivileged users. This allows them to import and convert container images.
  - `pyxis-sources`: installs a tarball with the Pyxis plugin source files. The tarball is used by `cm-wlm-setup` to compile the Pyxis plugin on a cluster. The version of the package (section 9.1), found using `yum info pyxis-sources` or `apt-cache show pyxis-sources`, implies the Pyxis source version that is provided. For example: `pyxis-sources-0.20.0-[...].rpm` provides the Pyxis source for version 0.20.0.
- The hardened packages that can be installed from the BCM repositories to replace the standard `enroot` and `enroot+caps` are:
  - `enroot-hardened`: provides hardened main utility and helper files.
  - `enroot-hardened+caps`: Provides extra capabilities to unprivileged users so that they can import and convert containers themselves.

The package `pyxis-sources` installs the tarball at `/cm/local/apps/slurm/var/pyxis/pyxis-sources.tar.gz`. If the administrator needs a version of Pyxis other than the one provided by the package, then the archive can be replaced with a source tarball of the same name.

How Are Enroot And Pyxis Set Up In BCM?

Pyxis and Enroot can be set up or removed by the administrator by choosing the appropriate options when Slurm is set up. In order to choose the appropriate options, `cm-wlm-setup` can be run in step-by-step mode (page 361). Alternatively, Pyxis-related command line options can be specified for the `cm-wlm-setup` arguments `--add-pyxis` and `--reinstall-pyxis`.

The step-by-step mode eventually presents a screen where the Pyxis setup can be enabled using a plugin:

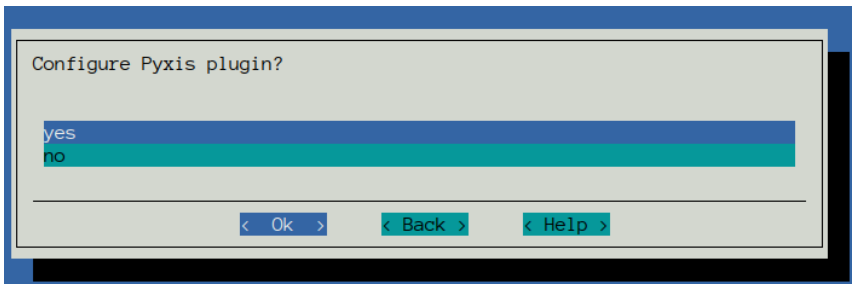


Figure 7.14: `cm-wlm-setup` Pyxis setup screen

The Pyxis screen is available for RHEL8-based systems and Ubuntu 20 and beyond. Older systems are not supported.

If the plugin is enabled in the Pyxis setup screen, then `cm-wlm-setup` installs the `enroot` and `enroot+caps` packages from `https://github.com/NVIDIA/enroot/releases` into the software images where the Slurm client role is to be assigned. The `cm-wlm-setup` uitility also installs them directly on the head node if the head node was selected to run jobs. Pyxis sources are downloaded from GitHub, compiled with the installed Slurm, and installed in the appropriate Slurm directory.

If the administrator enables the Pyxis plugin, then a new screen with Enroot settings is shown:

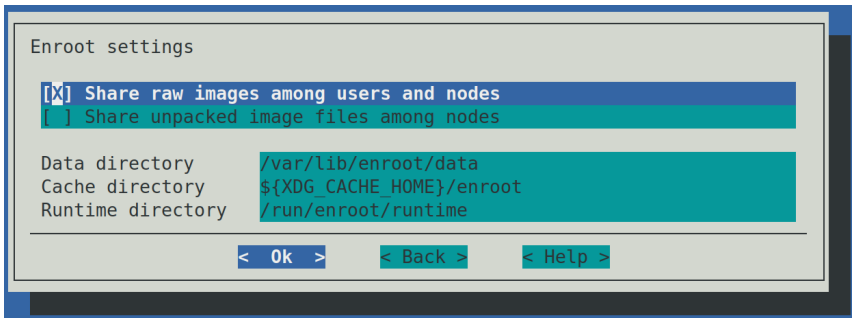


Figure 7.15: `cm-wlm-setup` Enroot settings screen

The Enroot settings have reasonable default values. The administrator can change these:

- `Share raw images among users and nodes`: If enabled, then the raw container image is shared among users and nodes, and the administrator must ensure that the cache directory is shared among all the compute nodes. Enabling the option disables the creation of a UID subdirectory by the Enroot prolog script.

  The following points about implementing Enroot cache sharing should be considered by the administrator:

  - Enabling the checkbox for the option normally changes the default value of the Enroot cache directory, as defined in the TUI in the `Cache directory` field, to `${XDG_CACHE_HOME}/enroot`. This field is normally evaluated by the Enroot installation to the home directory of the user:

    `~/.cache/enroot`

    when a job is executed.

    The default value can also be changed in the TUI to any shared directory. If the specified directory does not exist, then `cm-wlm-setup` creates it with Unix directory permissions set to `chmod 00777`, which allows all users to share container images in the cache with each other on all the nodes. The extra `0` prefix is a non-POSIX-compliant GNU/Linux extension that clears the SUID and SGID bits so that users do not get elevated privileges simply through sharing the images.
  - Restrictions can be placed on the ability of regular users to share images and to create files in the cache, by setting more restrictive permissions for the directory manually, after the `cm-wlm-setup` run.

    For example, the permissions can be set to 00770, while the ownership group of the directory is changed to a group with all the Pyxis users in it. In this case, only the users within the Pyxis users group can create the image layers in the cache.
  - An alternative to cache sharing of images among users and nodes is to save Squashfs images in a shared directory, outside the cache. Then the users can specify a full path to a shared Squashfs image on the cluster in the `srun/sbatch` command line. In this case there is no need to share the cache among the nodes, as the Squashfs images are not copied over to cache before execution.

  Only the administrator should have access to updating the image.

- `Share unpacked container images among nodes`: Enables sharing of unpacked container images (image filesystems) among nodes. A user job can modify the filesystem, so it is not recommended to share this directory among users.

  Enabling the option disables removal of the Enroot data directory, if the container name is specified. This allows a user to keep the container filesystem between jobs run by the user. It is up to the user or the administrator to clean up the data directory if the option is enabled.

  If the data directory is shared among nodes, then it is up to a user to ensure that the container filesystem is unpacked at least once before real jobs start. Otherwise, a race condition is possible when the images are extracted simultaneously on several nodes.

- `Cache directory`: Path to the cache directory where raw container image layers are stored.

  If the directory is not shared, then the Slurm prolog script creates a subdirectory with a name that is the job user ID, while epilog cleans up that subdirectory. If the directory is shared, then neither prolog nor epilog touches the directory.

- `Data directory`: Directory where the container filesystems (unpackaged images) are stored. If the directory is shared, then the epilog script checks if the container name is specified for the job, and skips removal of the container subdirectory.

- `Runtime directory`: A working directory with temporary files created by Enroot.

When Pyxis is set up, `cm-wlm-setup` also prepares the following configuration files for the compute nodes:

- `/etc/enroot/enroot.conf`: This is a symlink to `/cm/shared/apps/slurm/etc/enroot.conf`. The configuration file provides reasonable default settings that allows Enroot to be used by many users. Important settings in the file are:

  - `ENROOT_RUNTIME_PATH`: working directory for enroot, created per user. Default value: `/run/enroot/runtime/$(id -u)`
  - `ENROOT_CACHE_PATH`: directory where container layers are stored. Default value: `/run/enroot/cache/$(id -u)`
  - `ENROOT_DATA_PATH`: directory where the filesystems of running containers are stored. Default value: `/run/enroot/data/$(id -u)`
  - `ENROOT_SQUASH_OPTIONS`: options passed to `mksquashfs` to produce container images. Default value: `-noI -noD -noF -noX -no-duplicates`
  - `ENROOT_MOUNT_HOME`: mount the current user's home directory by default. Default value: `yes`.

  The administrator can change the symlink, or replace the file with a customized `enroot.conf`, if other values are preferred.

- `/etc/sysctl.d/80-enroot.conf`: symlink to `/cm/shared/apps/slurm/etc/enroot-sysctl.conf`. The file tunes `sysctl` parameters for Enroot.

- `/cm/local/apps/slurm/var/prologs/50-prolog-enroot.sh`: symlink to `/cm/shared/apps/slurm/prologs/prolog-enroot.sh`. This is the `slurmd` prolog that creates appropriate directories, with appropriate user permissions, that are used by Enroot.

- `/cm/local/apps/slurm/var/epilogs/50-epilog-enroot.sh`: symlink to `/cm/shared/apps/slurm/epilogs/epilog-enroot.sh`. Cleans the user directories used by Enroot.

When `cm-wlm-setup` finishes its Pyxis configuration run, then there is no need for the nodes to be rebooted. The plugin works immediately.

The new Slurm submission command option names start with either `--container` or `--no-container`. The full list of options can be displayed with the `--help` option. For example:

Example

```
[user@basecm11 ~]$ srun --help | grep container
      --container             Path to OCI container bundle
      --container-image=[USER@][REGISTRY#]IMAGE[:TAG]|PATH
                              [pyxis] the image to use for the container
      --container-mounts=SRC:DST[:FLAGS][,SRC:DST...]
                              [pyxis] bind mount[s] inside the container. Mount
      --container-workdir=PATH
                              [pyxis] working directory inside the container
      --container-name=NAME   [pyxis] name to use for saving and loading the
                              container on the host. Unnamed containers are
                              containers are not. If a container with this name
                              already exists, the existing container is used and
      --container-save=PATH   [pyxis] Save the container state to a squashfs
      --container-mount-home  [pyxis] bind mount the user's home directory.
      --no-container-mount-home
      --container-remap-root  [pyxis] ask to be remapped to root inside the
                              container. Does not grant elevated system
      --no-container-remap-root
                              [pyxis] do not remap to root inside the container
      --container-entrypoint  [pyxis] execute the entrypoint from the container
      --no-container-entrypoint
                              container image
      --container-writable    [pyxis] make the container filesystem writable
      --container-readonly    [pyxis] make the container filesystem read-only
```

Simple installation validation: The simplest way to validate the Pyxis/Enroot setup after Slurm setup is to try out an srun command:

Example

```
[user@basecm11 ~]$ module load slurm
[user@basecm11 ~]$ srun --container-image=ubuntu grep PRETTY /etc/os-release
pyxis: importing docker image: ubuntu
PRETTY_NAME="Ubuntu 24.04.2 LTS"
```

A more thorough installation validation: In order to perform a more thorough test of Pyxis/Enroot, an NCCL-based test can be used. NCCL is the NVIDIA Collective Communications Library (`https://docs.nvidia.com/deeplearning/nccl`), which is a library of multi-GPU collective communication primitives. The test can be found at `https://github.com/NVIDIA/nccl-tests`. A prebuilt container image, with the NCCL test already installed, can be started as shown in the example that follows.

It should be noted that running a multi-tenant cluster with all the export flags enabled as in the example may compromise security. It is therefore not recommended as a standard configuration.

Example

```
[user@basecm11 ~]$ module load slurm
[user@basecm11 ~]$ srun --export="NCCL_DEBUG=INFO,NCCL_IB_DISABLE=1,PMIX_MCA_gds=hash" -N 2 \
--ntasks-per-node=1 --gpus-per-task=1 --mpi=pmix --container-image=deepops/mpi-nccl-test \
/nccl_tests/build/all_reduce_perf -b 1M -e 4G -f 2 -g 1

pyxis: imported docker image: deepops/mpi-nccl-test
pyxis: imported docker image: deepops/mpi-nccl-test
# nThread 1 nGpus 1 minBytes 1048576 maxBytes 4294967296 step: 2(factor) warmup iters: 5 iters: 20 validation: 1
```

```
#
# Using devices
#   Rank  0 Pid 175945 on    node001 device  0 [0x00] Tesla V100-SXM3-32GB
#   Rank  1 Pid 180379 on    node002 device  0 [0x00] Tesla V100-SXM3-32GB
node001:175945:175945 [0] NCCL INFO Bootstrap : Using ens3:10.141.0.5<0>
node001:175945:175945 [0] NCCL INFO NET/Plugin : No plugin found (libnccl-net.so), using internal implementation
node001:175945:175945 [0] NCCL INFO NCCL_IB_DISABLE set by environment to 1.
node001:175945:175945 [0] NCCL INFO NET/Socket : Using [0]ens3:10.141.0.5<0>
node001:175945:175945 [0] NCCL INFO Using network Socket
NCCL version 2.11.4+cuda11.6
node002:180379:180379 [0] NCCL INFO Bootstrap : Using ens3:10.141.0.6<0>
node002:180379:180379 [0] NCCL INFO NET/Plugin : No plugin found (libnccl-net.so), using internal implementation
node002:180379:180379 [0] NCCL INFO NCCL_IB_DISABLE set by environment to 1.
node002:180379:180379 [0] NCCL INFO NET/Socket : Using [0]ens3:10.141.0.6<0>
node002:180379:180379 [0] NCCL INFO Using network Socket
node002:180379:182573 [0] NCCL INFO Trees [0] -1/-1/-1->1->0 [1] 0/-1/-1->1->-1
node001:175945:177561 [0] NCCL INFO Channel 00/02 :    0   1
node001:175945:177561 [0] NCCL INFO Channel 01/02 :    0   1
node001:175945:177561 [0] NCCL INFO Trees [0] 1/-1/-1->0->-1 [1] -1/-1/-1->0->1
node001:175945:177561 [0] NCCL INFO Channel 00 : 1[60] -> 0[60] [receive] via NET/Socket/0
node002:180379:182573 [0] NCCL INFO Channel 00 : 0[60] -> 1[60] [receive] via NET/Socket/0
node001:175945:177561 [0] NCCL INFO Channel 01 : 1[60] -> 0[60] [receive] via NET/Socket/0
node002:180379:182573 [0] NCCL INFO Channel 01 : 0[60] -> 1[60] [receive] via NET/Socket/0
node001:175945:177561 [0] NCCL INFO Channel 00 : 0[60] -> 1[60] [send] via NET/Socket/0
node002:180379:182573 [0] NCCL INFO Channel 00 : 1[60] -> 0[60] [send] via NET/Socket/0
node001:175945:177561 [0] NCCL INFO Channel 01 : 0[60] -> 1[60] [send] via NET/Socket/0
node002:180379:182573 [0] NCCL INFO Channel 01 : 1[60] -> 0[60] [send] via NET/Socket/0
node001:175945:177561 [0] NCCL INFO Connected all rings
node002:180379:182573 [0] NCCL INFO Connected all rings
node001:175945:177561 [0] NCCL INFO Connected all trees
node001:175945:177561 [0] NCCL INFO threadThresholds 8/8/64 | 16/8/64 | 8/8/512
node001:175945:177561 [0] NCCL INFO 2 coll channels, 2 p2p channels, 1 p2p channels per peer
node002:180379:182573 [0] NCCL INFO Connected all trees
node002:180379:182573 [0] NCCL INFO threadThresholds 8/8/64 | 16/8/64 | 8/8/512
node002:180379:182573 [0] NCCL INFO 2 coll channels, 2 p2p channels, 1 p2p channels per peer
node001:175945:177561 [0] NCCL INFO comm 0x1551f0001000 rank 0 nranks 2 cudaDev 0 busId 60 - Init COMPLETE
#
#                                                     out-of-place                       in-place
#       size         count      type   redop     time   algbw   busbw  error     time   algbw   busbw  error
#        (B)    (elements)                       (us)  (GB/s)  (GB/s)            (us)  (GB/s)  (GB/s)
node001:175945:175945 [0] NCCL INFO Launch mode Parallel
node002:180379:182573 [0] NCCL INFO comm 0x1551f8001000 rank 1 nranks 2 cudaDev 0 busId 60 - Init COMPLETE
     1048576        262144     float     sum   1491.5    0.70    0.70  0e+00   1421.3    0.74    0.74  0e+00
     2097152        524288     float     sum   3077.5    0.68    0.68  0e+00   2568.8    0.82    0.82  0e+00
     4194304       1048576     float     sum   4617.3    0.91    0.91  0e+00   4622.5    0.91    0.91  0e+00
     8388608       2097152     float     sum   9483.1    0.88    0.88  0e+00   8911.9    0.94    0.94  0e+00
    16777216       4194304     float     sum    17516    0.96    0.96  0e+00    18613    0.90    0.90  0e+00
    33554432       8388608     float     sum    34799    0.96    0.96  0e+00    41837    0.80    0.80  0e+00
    67108864      16777216     float     sum    99790    0.67    0.67  0e+00    83126    0.81    0.81  0e+00
   134217728      33554432     float     sum   204614    0.66    0.66  0e+00   199530    0.67    0.67  0e+00
   268435456      67108864     float     sum   319701    0.84    0.84  0e+00   341630    0.79    0.79  0e+00
   536870912     134217728     float     sum   608809    0.88    0.88  0e+00   683016    0.79    0.79  0e+00
  1073741824     268435456     float     sum  1337187    0.80    0.80  0e+00  1247369    0.86    0.86  0e+00
  2147483648     536870912     float     sum  2638741    0.81    0.81  0e+00  2743454    0.78    0.78  0e+00
  4294967296    1073741824     float     sum  5996381    0.72    0.72  0e+00  5430549    0.79    0.79  0e+00
```

```
# Out of bounds values : 0 OK
# Avg bus bandwidth    : 0.810575
#
```

The test demonstrates the usage of PMIX, MPI, and GPU in Enroot containers on multiple nodes. In the preceding example, 2 nodes with 1 GPU on each is requested by the job. For better results the cluster administrator can tune the test parameters.

It should be noted that the image is quite large, and requires enough free space under `/var`. Also, the transfer timeout (`ENROOT_TRANSFER_TIMEOUT`) in `enroot.conf` must be large enough to download such a large container image to the compute nodes. A value of at least 600 seconds is recommended.

If there are issues when executing MPI jobs with PMIX, then to help debug the issues the Pyxis documentation (`https://github.com/NVIDIA/pyxis/wiki/Setup`) suggests setting up the following environment variables:

- `PMIX_MCA_ptl=^usock`
- `PMIX_MCA_psec=none`
- `PMIX_SYSTEM_TMPDIR=/var/empty`
- `PMIX_MCA_gds=hash` (as configured on page 369)

This configuration change is typically carried out in the software image of the regular nodes. For example, for a node category called `default`:

```
[root@basecm11 ~]# category=default
[root@basecm11 ~]# cat <<EOF >> /cm/images/${category}/etc/default/slurmd
PMIX_MCA_ptl=^usock
PMIX_MCA_psec=none
PMIX_SYSTEM_TMPDIR=/var/empty
PMIX_MCA_gds=hash
EOF
[root@basecm11 ~]# systemctl restart slurmd
[root@basecm11 ~]# cmsh -c "device; imageupdate -c ${category} -w"
```

7.3.4 Prolog And Epilog Scripts

What Prolog And Epilog Scripts Do

The workload manager runs prolog scripts before job execution, and epilog scripts after job execution. The purpose of these scripts can include:

- checking if a node is ready before submitting a job execution that may use it
- preparing a node in some way to handle the job execution
- cleaning up resources after job execution has ended.

The administrator can run custom prolog or epilog scripts for the queues from CMDaemon for LSF, by setting such scripts in the Base View or `cmsh` front ends.

Example

```
[basecm11->wlm[lsf]->jobqueue]% use normal
[basecm11->wlm[lsf]->jobqueue[normal]]% show | grep -i epilog
Prolog/Epilog user               root
Epilog
Host epilog                      /cm/local/apps/cmd/scripts/epilog
```

For Slurm, there are global prolog and epilog scripts, but editing them is not recommended. Indeed, in order to discourage editing them, the scripts cannot be set using the cluster manager front ends. Instead the scripts must be placed by the administrator in the software image, and the relevant nodes updated from the image.

Detailed Workings Of Prolog And Epilog Scripts

Even though it is not recommended, some administrators may nonetheless wish to link and edit the scripts directly for their own needs, outside of the Base View or `cmsh` front ends. A more detailed explanation of how the prolog scripts work therefore follows:

When a workload manager is configured using `cm-wlm-setup` or using the Base View setup wizard, then the workload manager is configured to run the generic prolog located in `/cm/local/apps/cmd/scripts/prolog`, and the generic epilog located in `/cm/local/apps/cmd/scripts/epilog`. The generic prolog and epilog scripts call a sequence of scripts for a particular workload manager in special directories. The directories have paths in the format:

1. `/cm/local/apps/<`*workload manager*`>/var/prologs/`
2. `/cm/local/apps/<`*workload manager*`>/var/epilogs/`

In these directories, scripts are stored with names that have suffixes and prefixes associated with them that make them run in special ways, as follows:

- **suffixes used in the prolog/epilog directory:**
  - `-prejob` script runs prior to all jobs
- **prefixes used in the prolog/epilog directory:**
  - `00-` to
  - `99-`

  Number prefixes determine the order of script execution, with scripts with a lower number running earlier.

The script names can therefore look like:

Example

- `01-prolog-prejob`
- `10-prolog-prejob`

Return values for the prolog/epilog scripts have these meanings:

- `0`: the next script in the directory is run.
- *A non-zero return value*: no further scripts are executed from the prolog/epilog directory.

Often, the script in a prolog/epilog directory is not a real script but a symlink, with the symlink going to a real file located in a different directory. The general script is then able to take care of what is expected of the symlink. The name of the symlink, and destination file, usually hints at what the script is expected to do.

For example, if any health checks are marked to run as prejob checks during `cm-wlm-setup` configuration, then Slurm uses the symlink `01-prolog-prejob` within the prolog directory `/cm/local/apps/<`*workload manager*`>/var/prologs/`. The symlink links to the script `/cm/local/apps/cmd/scripts/prolog-prejob`. In this case, the script is expected to run prior to the job.

Example

```
[root@basecm11 apps]# pwd
/cm/local/apps
[root@basecm11 apps]# ls -l *slurm*/var/prologs/
slurm/var/prologs/:
total 0
lrwxrwxrwx 1 root root ... 01-prolog-prejob -> /cm/local/apps/cmd/scripts/prolog-prejob
```

Epilog scripts (which run after a job run) have the location /cm/local/apps/<*workload manager*>/var/epilogs/. Epilog script names follow the same execution sequence pattern as prolog script names.

It should be noted that the 01-prolog-prejob symlink is created and removed by BCM on each compute node where prejob is enabled in the workload manager entity. Each such entity provides a Enable Prejob parameter that affects the symlink existence:

Example

```
[head->wlm[slurm]]% get enableprejob
yes
[head->wlm[slurm]]%
```

This parameter is set to yes by cm-wlm-setup when at least one health check is selected as a prejob one. If any healthcheck was configured as a prejob check before cm-wlm-setup execution, and the administrator had a checkmark for that health check, then the prejob is considered enabled.

Workload Manager Configuration For Prolog And Epilog Scripts

BCM configures generic prologs and epilogs during workload manager setup with cm-wlm-setup. The administrator can configure prologs and epilogs using appropriate parameters in the configuration of the workload managers, by creating the symlinks in the local prologs and epilogs directories.

Generic prologs and epilogs are configured by default to run on job compute nodes (one run per each node per job) for Slurm and LSF.

The following parameters for prologs and epilogs can be configured with cmsh or Base View:

- **Slurm**
  - Prolog Slurmctld: the fully qualified path of a program to execute before granting a new job allocation. The program is executed on the same node where the slurmserver role is assigned. The parth corresponds to the PrologSlurmctld parameter in slurm.conf.
  - Epilog Slurmctld: the fully qualified path of a program to execute upon termination of a job allocation. The program is executed on the same node where the slurmserver role is assigned. Corresponds with the EpilogSlurmctld parameter in slurm.conf.
  - Prolog: the fully qualified path of a program to execute on job compute nodes before granting a new job or step allocation. The program corresponds to the Prolog parameter, and by default points to the generic prolog. This prolog runs on every node of the job if the Prolog flags parameter contains the flag Alloc (the default value), otherwise it is executed only on the first node of the job.
  - Epilog: the fully qualified path of a program to execute on job compute nodes when the job allocation is released.
- **LSF**
  - Prolog: the fully qualified path of a program to execute on the LSF server node on job allocation. As an LSF queue parameter, it corresponds to the PRE_EXEC parameter in lsb.queues.
  - Epilog: the fully qualified path of a program to execute on the LSF server node on job allocation release. As an LSF queue parameter, it corresponds to the POST_EXEC parameter in lsb.queues
  - Host prolog: the fully qualified path of a program to execute on each node of a job before the job is started. Corresponds to the HOST_PRE_EXEC parameter in lsb.queues. By default it is configured to run the generic prolog
  - Host epilog: the fully qualified path of a program to execute on each node of a job after the job is finished. Corresponds to the HOST_POST_EXEC parameter in lsb.queues. By default it is configured to run the generic epilog.

7.4 Enabling, Disabling, And Monitoring Workload Managers

Enabling And Disabling A WLM

A WLM can be disabled for all nodes with `cm-wlm-setup`. Disabling the WLM means the workload management services are stopped by removing roles, and removing the WLM cluster object.

Alternatively, a WLM can be enabled or disabled by the administrator using role addition and role removal with Base View or `cmsh`. This is described further on in this section.

Multiple WLM instances of the same type: Versions of NVIDIA Base Command Manager prior to 9.0 already had the ability to have different workload managers run at the same time. However, NVIDIA Base Command Manager version 9.0 introduced the additional ability to run many workload managers of the same kind at the same time.

Example

Two WLM instances, Slurm and LSF, are already running at the same time in the cluster, with each WLM assigned to one category. Then, BCM can start up a third WLM instance, such as another Slurm WLM instance. These WLM instances are alternatively called WLM clusters, because they effectively allow one cluster to function as many separate clusters as far as running WLMs is concerned.

From the Base View or `cmsh` point of view a WLM consists of

- a WLM server, usually on the head node
- WLM clients, usually on the compute nodes

For the administrator, enabling or disabling the servers or clients is then simply a matter of assigning or unassigning a particular WLM server or client role on the head or compute nodes, as deemed appropriate.

The administrator typically also sets up an appropriate WLM environment module (`slurm`, `lsf`), so that it is loaded up for the end user (section 2.2.3).

7.4.1 Enabling And Disabling A WLM With Base View

A particular WLM package may be installed, but the WLM may not be enabled. This can happen, for example, if disabling a WLM that was previously enabled.

If a WLM instance exists, then the WLM client, submission, and server roles can be enabled or disabled from Base View by assigning or removing the appropriate roles to nodes, categories, or configuration overlays. Within the role, the properties of the WLM may be further configured by setting options.

Workload Manager Role Assignment To An Individual Node With Base View

Workload Manager Server The following roles are WLM roles that can be assigned to a node:

- `server`
- `submit`
- `accounting` (for the Slurm WLM only, to configure and run the `slurmdbd` service)
- `client`

Typically these roles are assigned using configuration overlays.

For example, if the default Slurm configuration is carried out with `cm-wlm-setup`, then the Slurm server role is assigned to the head node, `basecm11`, which can be viewed using the navigation path:

`Devices > All Devices > Head Node` `> Settings > Roles` `> slurmserver`

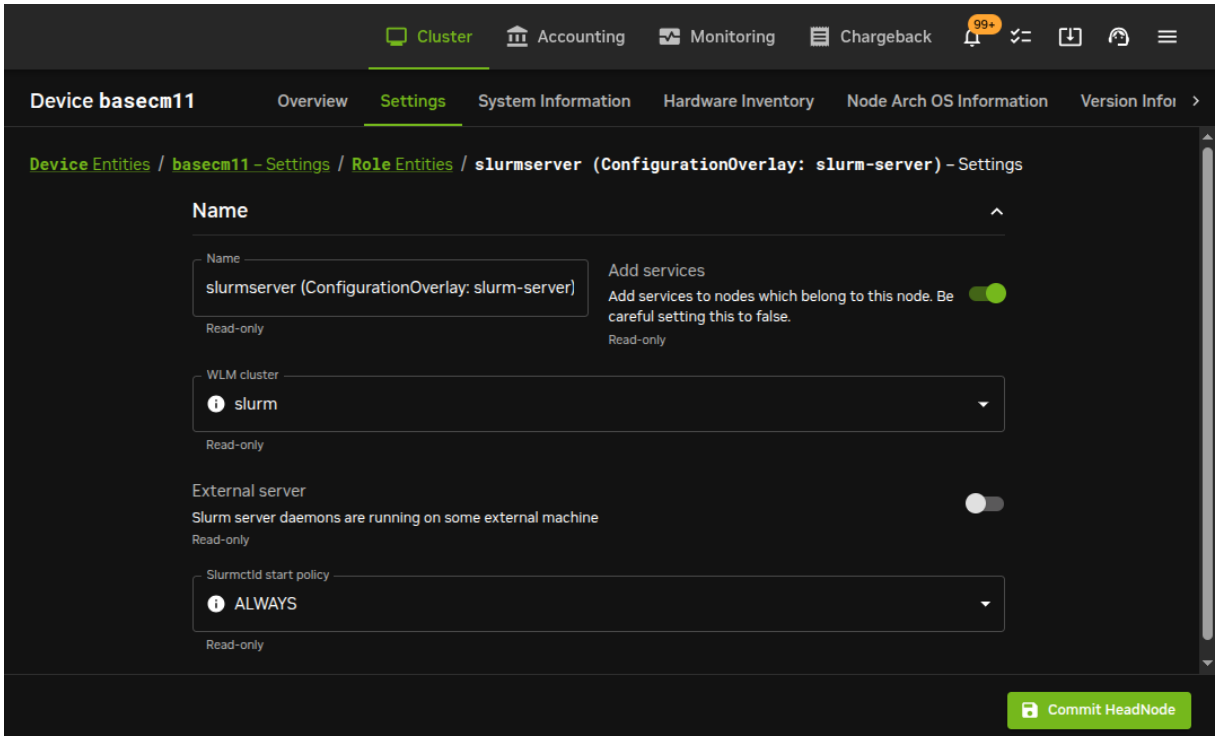


Figure 7.16: Workload management role assignment on a head node

Overriding the configuration overlay assignment is possible, at node or category level. For example at node level the role can be defined by assignment at node level, using the navigation path:

`Devices > All Devices > Head Node` > `Settings > Roles` > `+Add > SlurmServerRole`

This allows parameters to be changed within the role. For example, for the `SlurmServerRole` the `Slurmctld start policy` can then be modified at node level.

Workload manager options that are not regarded as server role options can be found under the workload manager object. For Slurm this is in the navigation path:

`HPC > Workload Management Clusters`

For example, for a Slurm instance in the Workload Management Clusters pane, a `builtin` or `backfill` option can be set for the `Scheduler` parameter (figure 7.17).

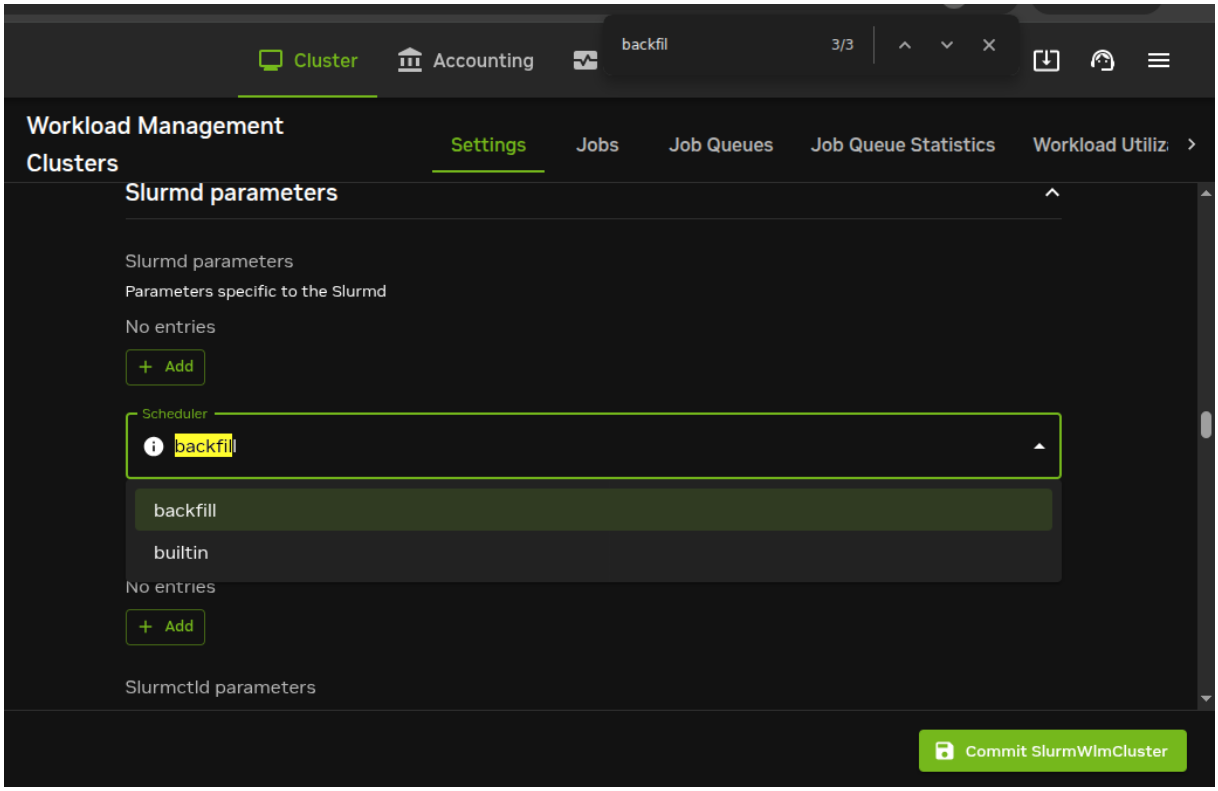


Figure 7.17: Workload management role assignment options on a head node

To have the server start up on non-head nodes (but not for a head node), the `imageupdate` command (section 5.6.2) can be run. The workload manager server process and any associated schedulers then automatically start up.

Workload Manager Client Similarly, the workload manager client process can be enabled on a node or head node by having the workload manager client role assigned to it at a node level. Some basic options can be set for the client role.

Saving the role, and then running `imageupdate` (section 5.6.2), automatically starts up the client process with the options chosen, and managed by CMDaemon.

Workload Manager Role Assignment To A Category With Base View

It is true that workload manager role assignment can be done as described in the preceding text for individual non-head nodes. However it is usually more efficient to assign roles using categories or configuration overlays, due to the large number of compute nodes in typical clusters.

For example, the case can be considered of all physical on-premises non-head nodes. By default these are in the `default` category. This means that, by default, roles in the category are automatically assigned to all those non-head nodes, unless, as an exception, an individual node configuration overrides the category setting and uses a role setting instead at node level.

The workload manager roles for the category `default` can be viewed by using the navigation path:

`Grouping > Categories[default]` > `Settings > Roles`

If the role has been assigned at category level, then clicking through to the subpane for a role allows role options to be edited and saved.

However,the default configuration using `cm-wlm-setup` assigns the role at configuration overlay level, which means that the role options (role settings) can

- either be configured using the navigation path from the configuration overlay level:

  `Grouping > Configuration Overlays[`*configuration overlay role*`] > Settings > Roles >` *role under configuration overlay role* `> Role settings`

- or they can be configured by adding a role at category level (or node level), with the `+Add` option. For example, for category level using the `default` category:

  `Grouping > Categories[default] > Settings > Roles > + Add >` *category-level role*

For compute nodes, the role assigned is usually a workload manager client. If the assigned role is that of a workload manager client, then the node with that role can have queues, GPUs, and other parameters specified for it.

For example, queues can then be assigned using the navigation path:

`HPC > WLM Management Clusters[`*Slurm instance*`] > Job Queues`

while GPUs, if using Slurm as the workload manager with default settings, with a default `cm-wlm-setup` installation, have a configuration overlay `slurm-client-gpu` that is specified using the navigation path:

`Grouping > Configuration Overlays > slurm-client-gpu > Settings > roles > slurmclient > Generic Resources > gpu`

The workload manager server role can also be assigned to a non-head node. For example, a Slurm server role can be taken on by a non-head node. This is the equivalent to the `--server-nodes` option of `cm-wlm-setup`.

Saving the roles with their options and then running `imageupdate` (section 5.6.2) automatically starts up the newly-configured workload manager.

Workload Manager Role Options With Base View

Each workload manager client role (sometimes loosely called a compute node role) has options that can be set for `GPUs`, `Queues`, and `Slots`. Generally, the value that is set for `Slots` is the number of jobs expected to run on a node simultaneously. This number can, for example, be set to the number of threads. Threads (virtual cores) in the x86_64 architecture are provided by Intel's hyper-threading (HT), or by AMD's simultaneous multithreading (SMT).

The physical CPU, the cores on the CPU, and the threads of a core (HT, SMT) should not be confused with each other. They are distinct concepts, and can all have different values.

- `Slots`, in a workload manager, corresponds in BCM to:
  - the `CPUs` setting (a `NodeName` parameter) in Slurm's `slurm.conf`
  - the `-n` flag, or the total number of processor slots requested, in IBM Spectrum LSF
  - the `nproc` (or `ncpus`) setting in PBS,

  In LSF setting the number of slots for the client role to `0` means that the client node does not run jobs on itself, but becomes a submit host, which means it is able to forward jobs to other client nodes.

  The default value for `Slots` is `AUTO`, which means that the value for `Slots` is auto-detected. The parameter may be alternatively be set to a non-negative number. Each WLM has a different implementation on how this is done. For instance, for Slurm, BCM uses Slurm's own auto-detection implementation.

- `Queues` with a specified name are available in their associated role after they are created. The creation of queues is described in sections 7.6.2 (using Base View) and 7.7.3 (using `cmsh`).

All server roles also provide the option to enable or disable the `External Server` setting. Enabling that means that the server is no longer managed by BCM, but provided by an external device.

The `cmsh` equivalent of enabling an external server is described on page 382.

7.4.2 Enabling And Disabling A Workload Manager With `cmsh`

A particular workload manager package may be set up, but not enabled. This can happen, for example, if no WLM server or WLM client role has been assigned.

If a WLM instance exists, then the WLM client, server, or submit roles can be enabled from `cmsh` by assigning it from within the `roles` submode. Within the assigned role, the properties of the WLM may be further configured by setting options.

Workload Manager Role Assignment To A Configuration Overlay With `cmsh`

In `cmsh`, workload manager role assignment to a configuration overlay (section 2.1.5) can be done using `configurationoverlay` mode. By default `cm-wlm-setup` run as a TUI session creates some configuration overlays with suggestive names, and assigns roles to the configuration overlays according to what the names suggest. Thus, for example, with the `cm-wlm-setup` TUI session used to carry out an express setup for Slurm, the configuration overlays that get created are the following:

Example

```
[basecm11->configurationoverlay]% list
Name (key)           Priority   All head nodes Nodes            Categories       Roles
-------------------- ---------- -------------- ---------------- ---------------- ----------------
slurm-accounting     500        yes                                              slurmaccounting
slurm-client         500        no                               default          slurmclient
slurm-server         500        yes                                              slurmserver
slurm-submit         500        no                               default          slurmsubmit
wlm-headnode-submit  600        yes                                              slurmsubmit
```

Nodes in the default category can take on the `slurmclient` or `slurmsubmit` role by setting the nodes for the role using the associated configuration overlays `slurm-client` or `slurm-submit`.

The `wlm-headnode-submit` configuration overlay is a special overlay. It is applied only to the head node, and is shared among all installed workload managers. Setting this overlay means that the head node, by default, has a submit role for a given workload manager.

Example

```
[basecm11->configurationoverlay]% use slurm-client
[basecm11->configurationoverlay[slurm-client]]% show
Parameter                      Value
------------------------------ ------------------------------------------------
Name                           slurm-client
Revision
All head nodes                 no
Priority                       500
Nodes
Categories                     default
Roles                          slurmclient
Customizations                 <0 in submode>
[basecm11->configurationoverlay[slurm-client]]% set nodes
node001  node002  node003  basecm11
[basecm11->configurationoverlay[slurm-client]]% set nodes node001..node002
[basecm11->configurationoverlay*[slurm-client*]]% commit
[basecm11->configurationoverlay[slurm-client]]% list
Name (key)           Priority   All head nodes Nodes            Categories       Roles
```

```
------------------- ---------- ------------- --------------- --------------- ----------------
slurm-accounting    500        yes                                           slurmaccounting
slurm-client        500        no            node001,node002 default         slurmclient
slurm-server        500        yes                                           slurmserver
slurm-submit        500        no                            default         slurmsubmit
wlm-headnode-submit 600        yes                                           slurmsubmit
[basecm11->configurationoverlay[slurm-client]]%
```

All the head nodes can also be made to take on the configuration overlay role by setting its `All head nodes` value to yes. The union set of `All head nodes` with `Nodes` is the set of nodes to which the role is applied for that configuration overlay.

Values for the parameters in a role, such as the `slurmclient` role, can be set within the configuration overlay:

Example

```
[basecm11->configurationoverlay[slurm-client]]% roles
[basecm11->configurationoverlay[slurm-client]->roles]% use slurmclient
[basecm11->configurationoverlay[slurm-client]->roles[slurmclient]]% show
Parameter                        Value
-------------------------------- ------------------------------------
Name                             slurmclient
Revision
Type                             SlurmClientRole
Add services                     yes
WLM cluster                      slurm
Slots                            0
All queues                       no
Queues                           defq
Features
Sockets                          0
Cores per socket                 0
Threads per core                 0
Boards                           0
Sockets per board                0
Real memory                      0B
Node address
Weight                           0
Port                             0
Tmp disk                         0
Reason
CPU spec list
Core spec count                  0
Mem spec limit                   0B
GPU auto detect                  BCM
Node customizations              <0 in submode>
Generic resources                <0 in submode>
Cpu bindings                     None
Slurm hardware probe auto detect yes
Memory autodetection slack       10.0%
IMEX                             no
Write procs always               no
Write only Procs                 no
Add to topology                  yes
Nodesets
Power profiles                   <submode>
```

```
Nodeset features
[basecm11->configurationoverlay[slurm-client]->roles[slurmclient]]%
```

After the workload manager roles are assigned or unassigned, and after running `imageupdate` (section 5.6.2) for non-head nodes, the associated workload manager services automatically start up or stop as appropriate.

The configuration overlay role values are inherited by categories and nodes, unless the categories and nodes have their own values set. Thus, for role properties, a value set at node level overrides values set at category level, and a value set at configuration overlay level overrides a value set at category level. This is typical of how properties of objects are inherited in BCM levels.

Workload Manager Role Assignment To A Category With `cmsh`

In `cmsh`, workload manager role assignment to a node category can be done using `category` mode, using the category name, assigning a role from the `roles` submode, setting the WLM instance for that role, and committing the modified role:

Example

```
[root@basecm11 ~]# cmsh
[basecm11]% category
[basecm11->category]% use default
[basecm11->category[default]]% roles
[basecm11->category[default]->roles]% assign slurmclient
[basecm11->category[default]->roles*[slurmclient*]]% wlm list
Type    Name (key)            Server nodes Submit nodes     Client nodes
------- --------------------- ------------ ---------------- ----------------
slurm   slurm1                basecm11     basecm11,node001 node001,node002
[basecm11->category[default]->roles*[slurmclient*]]% set wlmcluster slurm1
[basecm11->category[default]->roles*[slurmclient*]]% commit
```

Settings that are assigned in the `slurmclient` role of the category overrule the `slurmclient` role configuration overlay settings.

The role assignment at category level requires the value for a WLM instance to be specified for `wlmcluster` before the `commit` command is successful.

Workload Manager Role Assignment To An Individual Node With `cmsh`

In `cmsh`, assigning a workload manager role to a head node can be done in `device` mode. This can be done by using the head node name as the device, assigning the workload manager role to the device, setting the WLM instance value to the role within the `role` submode, and committing the modified role.

Example

```
[root@basecm11 ~]# cmsh
[basecm11]% device
[basecm11->device]% use basecm11
[basecm11->device[basecm11]]% roles
[basecm11->device[basecm11]->roles]% assign slurmserver
[basecm11->category[default]->roles*[slurmserver*]]% wlm list
Type    Name (key)            Server nodes Submit nodes     Client nodes
------- --------------------- ------------ ---------------- ----------------
slurm   slurm1                basecm11     basecm11,node001 node001,node002
[basecm11->category[default]->roles*[slurmserver*]]% set wlmcluster slurm1
[basecm11->device*[basecm11*]->roles*[slurmserver*]]% commit
[basecm11->device[basecm11]->roles[slurmserver]]%
```

For regular nodes, role assignment is done using `device` mode, using the node name. Th node name is assigned the workload manager role, the WLM instance value is set for that role in the `role` submode, and the modified role is committed.

Example

```
[root@basecm11 ~]# cmsh
[basecm11]% device
[basecm11->device]% use node001
[basecm11->device[node001]]% roles
[basecm11->device[node001]->roles]% assign slurmclient
[basecm11->device[node001]->roles*[slurmclient*]]% wlm list
Type    Name (key)            Server nodes Submit nodes     Client nodes
------- --------------------- ------------ ---------------- ----------------
slurm   slurm1                basecm11     basecm11,node001 node001,node002
[basecm11->device[node001]->roles*[slurmclient*]]% set wlmcluster slurm1
[basecm11->device[node001]->roles*[slurmclient*]]% commit
[basecm11->device[node001]->roles[slurmclient]]%
```

The role assignment at node level requires the value for a WLM instance to be specified for `wlmcluster` before the `commit` command is successful.

Role assignment values set in `device` mode have precedence over any role assignment values set in `category` mode for that node. This means, for example, that if a node is originally in a node category with a `slurmclient` role and queues set, then when the node is assigned a `slurmclient` role from `device` mode, its queue properties are empty by default.

Setting Options For Workload Manager Settings With `cmsh`

In the preceding text, it is explained how the workload manager client or server is assigned a role (such as `slurmclient` or `slurmserver`) within the `roles` submode. It is done from within a main mode of `cmsh`. The main modes from which role assignment can be done are: `configurationoverlay`, `category` or `device`.

Options for workload managers in general: Whatever main mode is used, the workload manager options for a role can then be set with the usual object commands introduced in section 2.5.3.

- **WLM client options:** For example, the configuration options of a WLM client, such as the Slurm client, can be seen by using the `show` command for the role.

  Here it can be seen in the `slurmclient` role, provided by the default configuration overlay `slurm-client`:

  Example

  ```
  [basecm11->configurationoverlay[slurm-client]->roles[slurmclient]]% show
  Parameter                        Value
  -------------------------------- ------------------------------------------------
  Name                             slurmclient
  Revision
  Type                             SlurmClientRole
  Add services                     yes
  WLM cluster                      slurm
  Slots                            0
  All queues                       no
  Queues                           defq
  Features
  Sockets                          0
  ```

```
Cores per socket                  0
Threads per core                  0
Boards                            0
Sockets per board                 0
Real memory                       0B
Node address
Weight                            0
Port                              0
Tmp disk                          0
Reason
CPU spec list
Core spec count                   0
Mem spec limit                    0B
GPU auto detect                   BCM
Node customizations               <0 in submode>
Generic resources                 <0 in submode>
Cpu bindings                      None
Slurm hardware probe auto detect  yes
Memory autodetection slack        10.0%
IMEX                              no
Write procs always                no
Write only Procs                  no
Add to topology                   yes
Nodesets
Power profiles                    <submode>
Nodeset features
```

The `Slots` option can be set in the role

Example

```
[basecm11->configurationoverlay[slurm-client]->roles[slurmclient]]% set slots 2
[basecm11->configurationoverlay*[slurm-client*]->roles*[slurmclient*]]% commit
[basecm11->configurationoverlay[slurm-client]->roles[slurmclient]]%
```

- **WLM server options:** Similarly, WLM server options can be managed from an assigned server role. For Slurm, the `slurmserver` role for a device shows:

Example

```
[basecm11->configurationoverlay[slurm-server]->roles[slurmserver]]% show
Parameter                      Value
------------------------------ ------------------------------------------------
Name                           slurmserver
Revision
Type                           SlurmServerRole
Add services                   yes
WLM cluster                    slurm
External server                no
Slurmctld start policy         ALWAYS
```

Option to set an external workload manager: A workload manager can be set to run as an external server from its `slurmserver` role, within a configuration overlay or within device mode :

Example

```
[basecm11->device[basecm11]->roles[slurmserver]]% set externalserver on
[basecm11->device[basecm11]->roles[slurmserver*]]% commit
```

For convenience, setting the external server on the head node is recommended.

The Base View equivalent of configuring `externalserver` is described on page 378.

7.4.3 Monitoring The State Of Workload Manager Services

By default, the workload manager services are monitored. BCM attempts to restart the services using the service tools (section 3.15), unless the role for that workload manager service is disabled, or the service has been stopped.

Workload manager roles and corresponding services can be disabled using `cm-wlm-setup` (section 7.3), Base View role configuration (section 7.4.1), or `cmsh` role configuration (section 7.4.2).

The daemon service states can be viewed for each node using the shell, `cmsh`, or Base View (section 3.15).

Queue submission and scheduling daemons normally run on the head node.

Viewing Services In Base View

From Base View services are viewable using the navigation path to the services running on the node. For example, on a head node `basecm11`, the navigation path:

`Devices > All Devices > basecm11` > `Settings > Services`

leads to a display similar to figure 7.18:

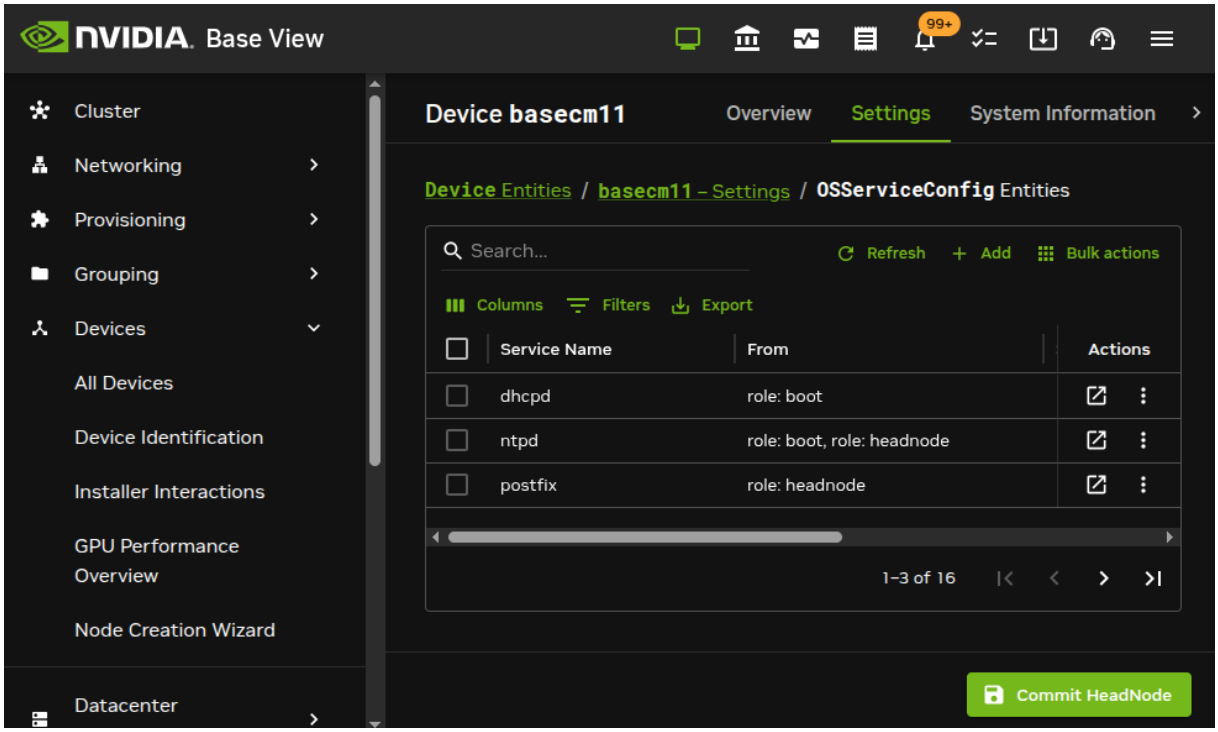


Figure 7.18: Services seen on head node in Base View

For a regular node, a similar navigation path for `node001`, for example, is:

`Devices > All Devices > node001` > `Settings > Services`

and leads to a display similar to figure 7.19:

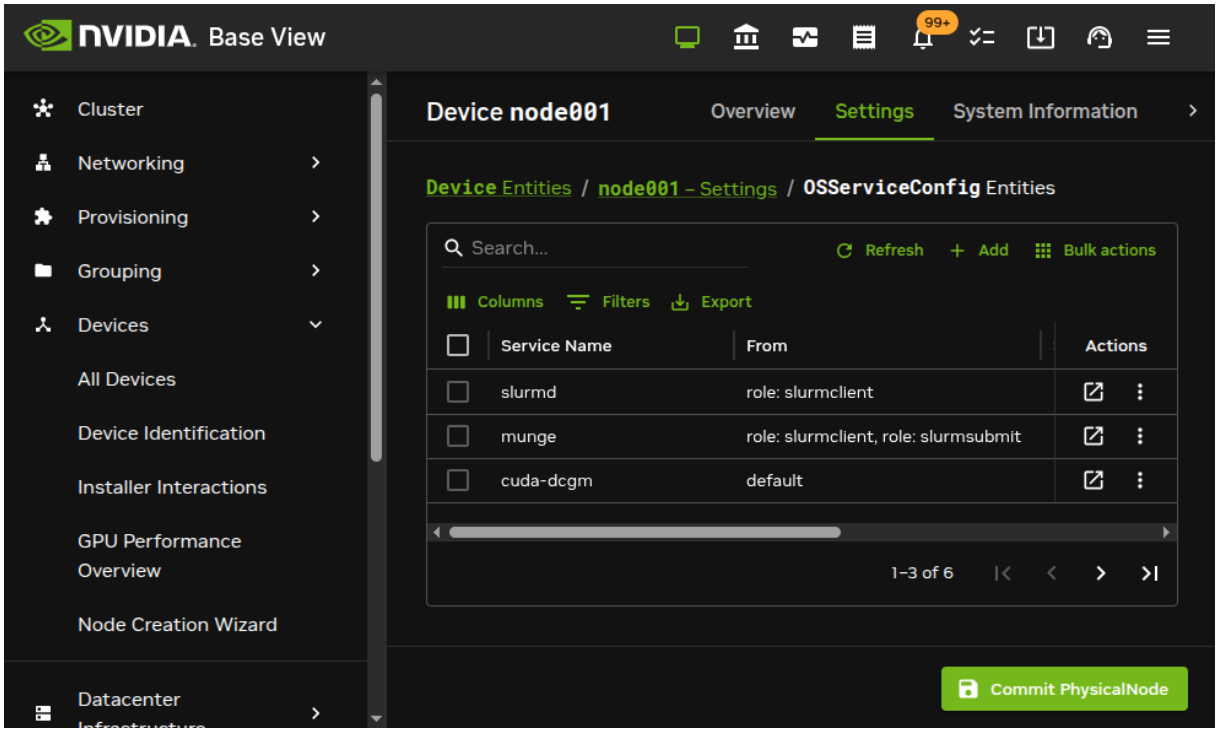


Figure 7.19: Services seen on regular node in Base View

Viewing Roles In Base View

Services are often set with BCM roles (section 2.1.5). The navigation path:

`Devices > All Devices > basecm11` > `Settings > Roles`

shows the list of roles. Figure 7.20 shows some of the Slurm roles that result in Slurm services running on the head node:

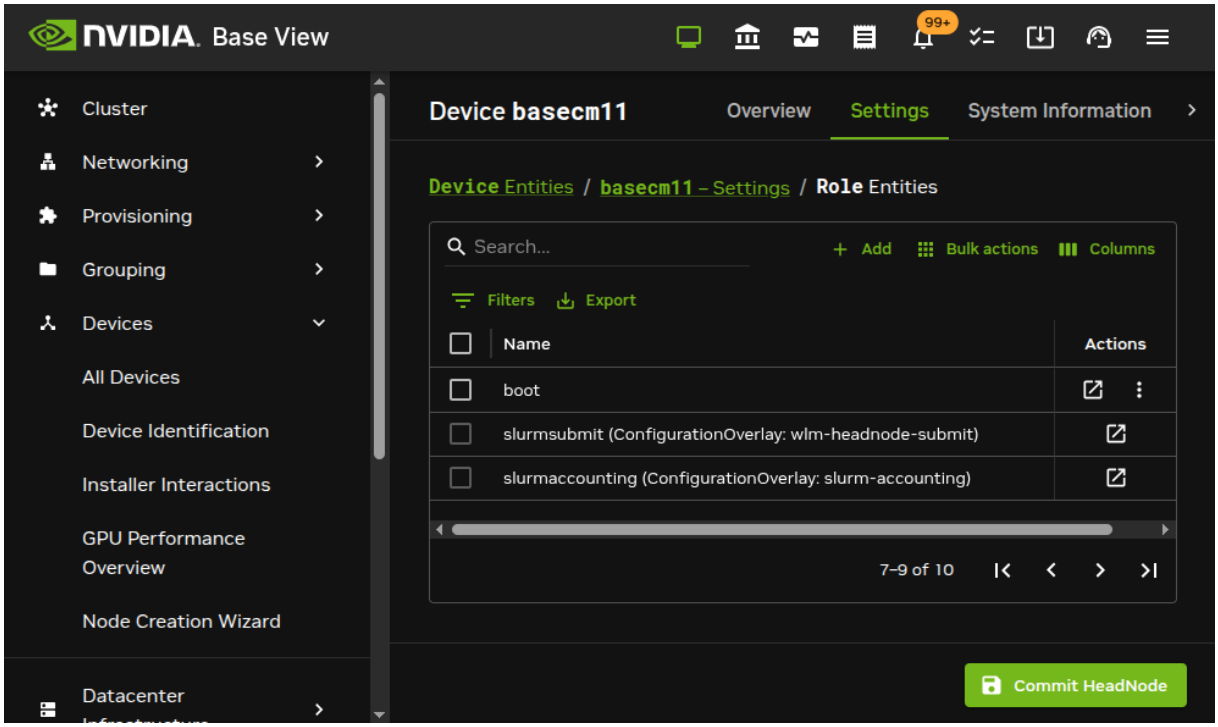


Figure 7.20: Roles seen on head node in Base View

Similarly, the navigation path:

`Devices > All Devices > node001` > `Settings > Roles`

shows the list of roles for a regular node. Figure 7.21 shows some of the Slurm roles that result in Slurm services running on the regular nodes:

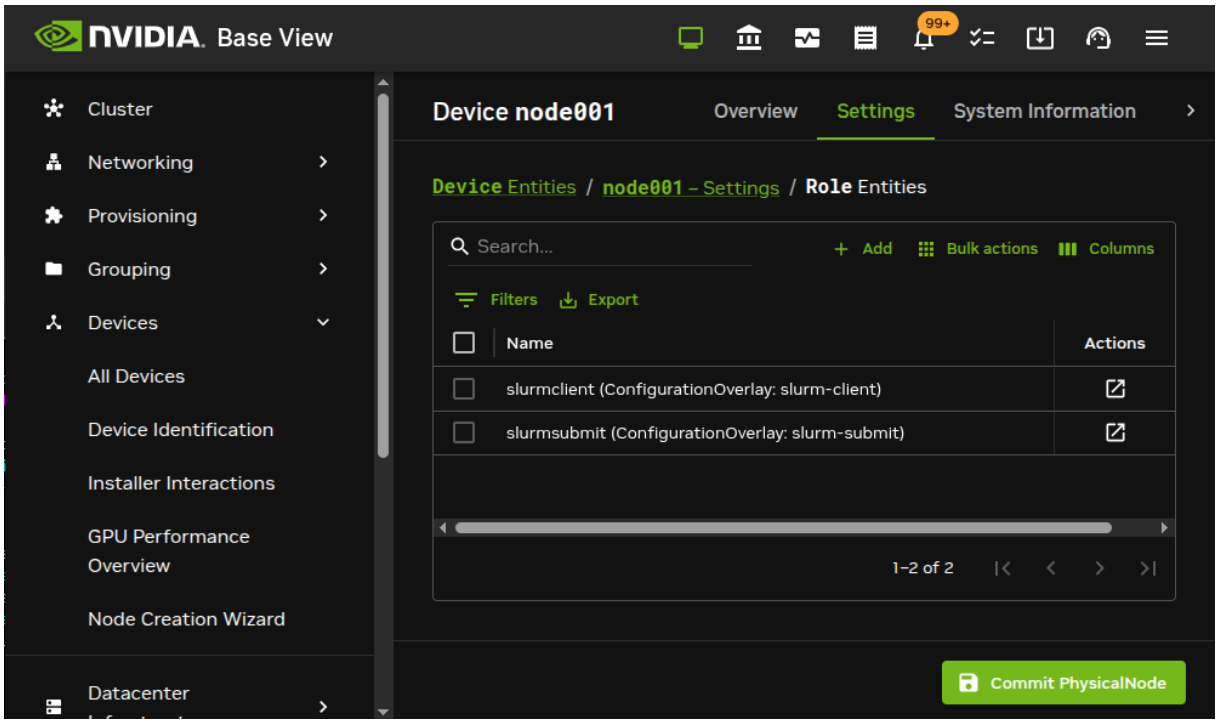


Figure 7.21: Roles seen on regular node in Base View

The roles seen in these figures are from the defaults that `cm-wlm-setup` provides in an express setup.

For regular nodes, the inheritance of roles from category level or configuration overlay level is indicated by the values in the parentheses alongside the role name. Thus, in figure 7.21, the `slurmsubmit` and `slurmclient` roles are decided by the default setting from the configuration overlay.

The assignment of roles can be varied to taste for WLMs. This allows WLM services to run on the head node or on the regular nodes.

Viewing Services And Roles In `cmsh`

From `cmsh` the services states are viewable from within `device` mode, using the `services` command. One-liners from the shell to illustrate this are (output elided):

Example

```
[root@basecm11 ~]# cmsh -c "device services node001; status"
Service      Status
------------ -----------
...
munge        [   UP   ]
named        [   UP   ]
nslcd        [   UP   ]
...
slurmd       [   UP   ]
[root@basecm11 ~]# cmsh -c "device services basecm11; status"
Service      Status
------------ -----------
...
munge        [   UP   ]
...
slurmctld    [   UP   ]
slurmdbd     [   UP   ]
```

Roles can be viewed from within the main modes of `configurationoverlay`, `category`, or `device`. One-liners to view these are:

Example

```
[root@basecm11 ~]# cmsh -c "configurationoverlay; list"
Name (key)           Priority   All head nodes Nodes            Categories       Roles
-------------------- ---------- -------------- ---------------- ---------------- ----------------
slurm-client         500        no                              default          slurmclient
slurm-server         500        yes                                              slurmserver
wlm-headnode-submit  600        yes                                              slurmsubmit

[root@basecm11 ~]# cmsh -c "category; use default; roles; list -p"
Name (key)
----------------------------
[overlay:slurm-client:500] slurmclient

[root@basecm11 ~]# cmsh -c "device use node001; roles; list -p"
Name (key)
---------------------------------------
[overlay:slurm-client:500] slurmclient


[root@basecm11 ~]# cmsh -c "device use basecm11; roles; list -p"
Name (key)
```

```
--------------------------------------
[750] backup
[750] boot
[750] firewall
[750] headnode
[750] monitoring
[750] provisioning
[750] storage
[overlay:slurm-server:500] slurmserver
```

The `-p|--priority` option displays of the `list` command the priority setting for the roles.

7.5 Configuring And Running Individual Workload Managers

BCM deals with the various choices of workload managers in as generic a way as possible. This means that not all features of a particular workload manager can be controlled, so that fine-tuning must be done through the workload manager configuration files. Workload manager configuration files that are controlled by BCM should normally not be changed directly because BCM overwrites them. However, overwriting by CMDaemon is prevented on setting the directive:

`FreezeChangesTo<`*workload manager*`>Config = <true|false>`

in `cmd.conf` (Appendix C), where <*workload manager*> takes the value of `Slurm` or `LSF`, as appropriate. The value of the directive defaults to `false`.

A list of configuration files that are changed by CMDaemon, the items changed, and the events causing such a change are listed in Appendix H.

A very short guide to some specific workload manager commands that can be used outside of the NVIDIA Base Command Manager 11 system is given in Appendix F.

7.5.1 Configuring And Running Slurm

Slurm Packages

At the time of writing of this section (July 2026), BCM is integrated with Slurm packages for Slurm versions 25.05, 25.11, and 26.05. Slurm version 25.11 is installed by default.

For Slurm version 25.05, the following packages are available from the BCM repositories for Ubuntu 24.04:

- `slurm25.05`: Simple Linux Utility for Resource Management, Slurm Workload Management.
- `slurm25.05-contribs`: contributed utilities for Slurm
- `slurm25.05-devel`: Development package for Slurm. Includes the header files and static libraries for the SLURM API.
- `slurm25.05-example-configs`: Example configuration files for Slurm.
- `slurm25.05-libpmi`: Slurm's implementation of the pmi libraries.
- `slurm25.05-openlava`: OpenLava/LSF wrappers for transition from OpenLava/LSF to Slurm.
- `slurm25.05-pam`: PAM module for restricting access to compute nodes via Slurm.
- `slurm25.05-perlapi`: Perl API to Slurm.
- `slurm25.05-prs`: Slurm PRS plugin.
- `slurm25.05-sackd`: Slurm authentication daemon. Used on login nodes that are not running slurmd daemons to allow authentication to the cluster.

- slurm25.05-slurmctld: Slurm control daemon.
- slurm25.05-slurmd: Slurm compute node daemon.
- slurm25.05-slurmdbd: Slurm database daemon.
- slurm25.05-slurmrestd: Slurm REST API translator.
- slurm25.05-torque: Torque/PBS wrappers for transition from Torque/PBS to Slurm.

For Slurm version 25.11, the value of 25.05 is simply replaced by 25.11 in the preceding list of packages.

Similarly, for Slurm version 26.05, the value of 25.05 is simply replaced by 26.05 in the preceding list of packages.

The distribution version of Slurm (package: slurm) is not integrated with BCM and conflicts with the preceding packages. It should not be used.

From BCM 11.34, BCM Slurm packages include the acct_gather_energy/ipmi plugin alongside the existing energy plugins (gpu, ibmaem, pm_counters, and rapl). This allows administrators to set AcctGatherEnergyType=acct_gather_energy/ipmi in slurm.conf to collect per-node power and energy data from IPMI-based baseboard sensors for job accounting and efficiency reporting. Configuration details can be found in the SchedMD documentation at https://slurm.schedmd.com/slurm.conf.html#OPT_AcctGatherEnergyType.

Important updates from upstream are patched into the BCM repositories. If updating Slurm packages, all the Slurm packages should be updated to the same version, on the compute nodes as well as on the scheduling node.

Building Custom Slurm Packages

The BCM Slurm packages can be rebuilt from source by the administrator, on the BCM cluster itself. Rebuilding allows important Slurm patches to be applied sooner than the standard BCM release cycle provides them, and allows the Slurm binaries to be optimized for a particular CPU or hardware configuration.

The build script, along with the RPM and DEB spec files that NVIDIA uses to build the packages, is shipped in a slurm<*version*>-builder package, for example slurm25.11-builder or slurm26.05-builder. Because the spec files are bundled in the package, no separate download of them is needed. The builder package is available from the BCM repositories, but is not installed by default. Installing it places the build script in /cm/local/apps/slurm/current/builder/, which is not in the standard $PATH.

Only Slurm version 25.11 and higher can be built with this script. The earlier Slurm version that BCM also ships, 25.05, cannot be built with it.

The script can be run on BCM clusters running RHEL 9, RHEL 10, Ubuntu 22.04, Ubuntu 24.04, or Ubuntu 26.04. SLES is not supported.

The script takes a required --version option, in *X.Y.Z* format:

```
[root@basecm11 ~]# cd /cm/local/apps/slurm/current/builder
[root@basecm11 builder]# ./create-packages.sh --version 25.11.5
```

The script downloads the Slurm source tarball and loads the modulefiles that are needed for the build. When the build finishes, the resulting packages are saved to /tmp/slurm<*version*>-<*revision*>-<*release-tag*>/, where <*revision*> is a build counter that increases with each run, and <*release-tag*> encodes the BCM version and a checksum of the source tarball.

Validating the build environment: Adding the --validate option carries out a dry run that checks the prerequisites without building anything. Each check is reported as [PASS] or [FAIL], and the script exits with 0 if every check passed, or 1 if any check failed:

```
[root@basecm11 builder]# ./create-packages.sh --version 25.11.5 --validate
Slurm version: 25.11.5                                                  [PASS]
Distribution                                                            [PASS]
Host tools                                                              [PASS]
Build tools                                                             [PASS]
Build dependencies                                                      [PASS]
Modules                                                                 [PASS]
Disk space                                                              [PASS]

All checks passed.
```

The checks cover the Slurm version, the distribution and its major version, the host tools (`wget`, `tar`, and `jq`), the build tool (`dpkg-buildpackage` on Ubuntu, or `rpmbuild` on RHEL), the build dependencies, the availability of the required modules, and whether at least 100MB of disk space is free.

Building inside a software image: Packages that are intended for a particular software image should be built inside that image, so that the build uses the same operating system, libraries, and modules that the image provides. The image should be cloned first, using `cmsh` or Base View, and the build then run inside the clone with `cm-chroot`:

```
[root@basecm11 ~]# cm-chroot /cm/images/slurm-builder \
/cm/local/apps/slurm/current/builder/create-packages.sh --version 25.11.5
```

The build dependencies are installed automatically, driven by a `dependencies.yaml` file that ships with the builder package, whether the script is run on the host or inside a `cm-chroot` image. No manual dependency installation is required beforehand.

Building from a different source tarball: A source tarball other than the one that the script downloads from upstream can be used by passing its path with `--archive`:

```
[root@basecm11 builder]# ./create-packages.sh --version 25.11.5 \
--archive /root/slurm-25.11.5.tar.bz2
```

The contents of the tarball must be inside a directory that matches the version being built. So, for Slurm 25.11.5, the tarball contents must be inside a directory called `slurm-25.11.5`, in the same way as the upstream Slurm tarballs.

The packages that are built are installed in the same way as for a Slurm upgrade, except that the packages are taken from the local directory instead of from the package repositories.

Upgrading From Earlier Slurm Versions To `slurm25.11`

Upgrading between major versions of Slurm is generally possible. At the time of writing (December 2025) of this section, the upstream Slurm policy allows upgrades from three previous major release versions since version 24.11. Details on the upgrade compatibility can be seen at the SchedMD site at `https://slurm.schedmd.com/upgrades.html#compatibility_window`.

It is a good idea to upgrade one version at a time, rather than jumping too many versions ahead, which can require a full wipe of the Slurm configuration.

If Slurm is using Pyxis (section 7.3.3), then upgrading the Slurm version means that Pyxis needs to be reinstalled using `cm-wlm-setup`. The reinstallation run for Pyxis compiles Pyxis and recreates a plugin directory for the new Slurm version under `/cm/local/apps/slurm/`.

An upgrade from one major version of Slurm to another can be carried out according to the following example, which is for an update from major version 25.05 to version 25.11, and avoids total reconfiguration of the Slurm configuration:

- It is recommended that no jobs are running. Draining nodes (section 7.7.4) is one way to arrange this over time. No new jobs run on a drained node, but old ones are allowed to finish.

- When all running jobs are finished, then Slurm server services—`slurmctld` and `slurmdbd`—should be stopped using `cmsh` or Base View (section 3.15.2):

  Example

```
[basecm11->device[basecm11]->services]% stop slurmctld
...Service slurmctld was stopped
[basecm11->device[basecm11]->services]% stop slurmdbd
...Service slurmdbd was stopped
```

- The old Slurm packages should then be removed. There can be only one version of Slurm at a time, so there will be a package installation conflict if a new version is installed while an old one is still there.

  Removal can be carried out on RHEL-based systems with, for example:

```
[root@basecm11 ~]# yum remove slurm25.05*
```

  In Ubuntu the equivalent is carried out using `apt remove slurm25.05*`.

  The old packages must also be removed from each software image that uses it:

  Example

```
[root@basecm11 ~]# cm-chroot-sw-img /cm/images/<software image>
...
        [root@<software image> /]# yum remove slurm25.05*
...removal takes place...
[root@<software image> /]# exit
```

  The `cm-chroot-sw-img` wrapper utility is discussed in section 9.4.1.

- The new packages can then be installed. For installation onto the RHEL head node, the installation might be carried out as follows:

```
[root@basecm11 ~]# yum install slurm25.11 slurm25.11-contribs slurm25.11-devel \
slurm25.11-libpmi slurm25.11-openlava slurm25.11-pam slurm25.11-perlapi \
slurm25.11-prs slurm25.11-sackd slurm25.11-slurmctld slurm25.11-slurmd \
slurm25.11-slurmdbd slurm25.11-slurmrestd slurm25.11-torque
```

  The same packages as on the head node are installed by default in the software image when installing with `cm-wlm-setup` in versions of Slurm from version 24 onward. The installation is carried out manually for the upgrade in each software image with:

  Example

```
[root@basecm11 ~]# cm-chroot-sw-img /cm/images/<software image>
...
[root@<software image> /]# yum install slurm25.11-slurmd slurm25.11-contribs slurm25.11-devel \
slurm25.11-libpmi slurm25.11-openlava slurm25.11-pam slurm25.11-perlapi \
slurm25.11-prs slurm25.11-sackd slurm25.11-slurmctld \
slurm25.11-slurmdbd slurm25.11-slurmrestd slurm25.11-torque
...installation takes place
[root@<software image> /]# exit
```

- The new Slurm version is then set in `cmsh` or Base View, in the Slurm WLM cluster configuration:

  Example

  ```
  [root@basecm11 ~]# cmsh
  [basecm11]% wlm use slurm
  [basecm11->wlm[slurm]]% set version 25.11; commit
  ```

- Slurm server services `slurmctld` and `slurmdbd` should then be started again using `cmsh` or Base View:

  Example

  ```
  [basecm11->device[basecm11]->services]% start slurmdbd
  [basecm11->device[basecm11]->services]% start slurmctld
  ```

- The nodes can then have their new image placed on them, and the new Slurm configuration can then be taken up. This can be done in the following two ways:

  1. The regular nodes can then be restarted to supply the live nodes with the new image and get the new Slurm configuration running.
  2. Alternatively, the `imageupdate` command (section 5.6.2) can be run on the live nodes to supply them with the image.

     Running the `imageupdate` command in dry mode (the default) first is recommended. The `synclog` command can then be run to check there are no unexpected changes that will take place due to the update. If all is well, then `imageupdate`'s wet mode flag `-w` can be used in order to really carry out the task.

     For example, the change can be checked, and then actually carried out, for the image on `node001` with:

     ```
     [root@basecm11 ~]# cmsh
     [basecm11]% device use node001
     [basecm11->device[node001]]% imageupdate
     Performing dry run (use synclog command to review result, then pass -w to perform real update)...
     ...some messages...
     imageupdate [ COMPLETED ]
     [basecm11->device[node001]]% synclog
     ...rsync dry run output...
     [basecm11->device[node001]]% imageupdate -w
     ...same messages as before, but this time it really happens...
     [basecm11->device[node001]]% commit
     ```

     The last `commit` command triggers the reconfiguration of the file `/etc/systemd/system/slurmd.service.d/99-cmd.conf` on the node. After a short time—around 30 seconds—the file is regenerated. The `slurmd` service on the node can then be restarted with:

     ```
     [root@basecm11 ~]# ssh node001 "systemctl daemon-reload"
     cmsh -c "device services node001; use slurmd; restart"
     ```

IMEX

NVIDIA IMEX (Internode Memory Exchange Service) is a secure service that facilitates the mapping of GPU memory over NVLink between the GPUs in an NVLIink domain.

The `imexctl` command in `device` mode lists the status and other properties of the nodes.

Example

```
[basecm11->device]% imexctl
Node                 IP             Status   Version     Connection        IP             Status
-------------------- -------------- -------- ----------- ----------------- -------------- ----------
dgx-gb200-c1         10.140.2.134   READY    570.172.08  dgx-gb200-m06-c1  10.140.2.134   CONNECTED
dgx-gb200-c1         10.140.2.134   READY    570.172.08  dgx-gb200-m06-c2  10.140.2.135   CONNECTED
dgx-gb200-c1-dpu0    0.0.0.0
dgx-gb200-c1-dpu1    0.0.0.0
dgx-gb200-c2         10.140.2.135   READY    570.172.08  dgx-gb200-m06-c1  10.140.2.134   CONNECTED
dgx-gb200-c2         10.140.2.135   READY    570.172.08  dgx-gb200-m06-c2  10.140.2.135   CONNECTED
basecm11             0.0.0.0        NONE
basecm11-k8s-01      0.0.0.0        NONE
basecm11-k8s-02      0.0.0.0        NONE
basecm11-k8s-03      0.0.0.0        NONE
basecm11-slogin-0    0.0.0.0        NONE
[basecm11->device]%
```

Pinning IMEX to a specific network (FQDN or IP in `nodes_config.cfg`): By default, the autogenerated IMEX peer list in `/etc/nvidia-imex/nodes_config.cfg` uses the short hostname of each node. IMEX can instead be pinned to a particular network. When a network has the `imex` parameter enabled, the fully qualified domain name (FQDN) on that network is written to `nodes_config.cfg` for each peer. For example, for an IMEX network called *imexnetwork*:

```
[root@basecm11 ~]# cmsh
[basecm11]% network use imexnetwork
[basecm11->network[imexnetwork]]% set -e imex yes
[basecm11->network*[imexnetwork*]]% commit
```

The resulting config file uses the FQDN for each node:

```
root@node001:~# cat /etc/nvidia-imex/nodes_config.cfg
# This section of this file was automatically generated by cmd. Do not edit manually!
# BEGIN AUTOGENERATED SECTION -- DO NOT REMOVE
node001.eth.cluster
node002.eth.cluster
node003.eth.cluster
# END AUTOGENERATED SECTION   -- DO NOT REMOVE
```

The peers can be written as IP addresses instead of FQDNs by also enabling the `imex ip` parameter on the same network:

```
[root@basecm11 ~]# cmsh
[basecm11]% network use imexnetwork
[basecm11->network[imexnetwork]]% set -e imex yes
[basecm11->network*[imexnetwork*]]% set -e "imex ip" yes
[basecm11->network*[imexnetwork*]]% commit
```

The resulting config file uses the IP address of each node on that network, with the node name retained as a comment:

```
root@node001:~# cat /etc/nvidia-imex/nodes_config.cfg
# This section of this file was automatically generated by cmd. Do not edit manually!
# BEGIN AUTOGENERATED SECTION -- DO NOT REMOVE
# node001
10.141.0.1
```

```
# node002
10.141.0.2
# node003
10.141.0.3
# END AUTOGENERATED SECTION   -- DO NOT REMOVE
```

For Slurm, BCM can enable the IMEX daemon either globally, or per job.

Enabling the IMEX daemon globally for Slurm: If IMEX is set globally, then it means that it is set to run on all nodes at runtime.

The advantage of this configuration is that it is easy to set up, and easy to test to see if it is all working.

A disadvantage is that a user running a job on one node can read memory from a job run by another user on another node. It requires overcoming difficult hurdles to carry out, but it is not impossible. The administrator should therefore weigh up if this seems a significant issue for the cluster being administered.

The IMEX global service can be run by setting the values for the service for the compute nodes. For example, nodes in a `dgx-gb200` category can have the service added and set to:

```
[basecm11->category[dgx-gb200]->services[nvidia-imex]]% show
Parameter                      Value
------------------------------ ------------------------------------------------
Revision
Service                        nvidia-imex
Run if                         always
Monitored                      yes
Autostart                      yes
Managed                        yes
```

Per-rack IMEX configuration can be forcibly created, regardless of domains and partitions, with:

```
[basecm11->category[dgx-gb200]->services[nvidia-imex]]% set -e rack yes
```

Enabling and disabling the IMEX daemon per job for Slurm: Configuring per job means that IMEX runs just before the job starts, and that the service runs only on a node that needs it for the job.

A disadvantage is that verifying that it actually works requires a job run.

Running the IMEX daemon per job has the advantage that one user running a job cannot read the memory of a job run by another user on another node.

To run IMEX per job, any global IMEX setting must first be cleared away. This can be done by first:

- removing the `nvidia-imex` service configuration entirely from CMDaemon

  Example

  ```
  [basecm11->category[dgx-gb200]->services[nvidia-imex]]% remove nvidia-imex; commit
  ```

- stopping the service on the compute nodes, for example with `pdssh` or `pdexec`, or simply carrying out a reboot

The value of `imex` in the `slurmclient` role can then be set:

```
[root@basecm11 ~]# cmsh
[basecm11]% configurationoverlay roles slurm-client-gpu
[basecm11->configurationoverlay[slurm-client-gpu]->roles]% use slurmclient
[basecm11->configurationoverlay[slurm-client-gpu]->roles[slurmclient]]% set imex yes
[basecm11->configurationoverlay*[slurm-client-gpu*]->roles*[slurmclient*]]% commit
[basecm11->configurationoverlay[slurm-client-gpu]->roles[slurmclient]]%
```

Committing that imex setting configures Slurm prolog and epilog scripts in the backend as follows:

- the prolog script configures the IMEX daemon with the nodes allocated to the GPU job and starts the IMEX daemon on the node
- the epilog cleans up the configuration and stops the IMEX daemon on the node

The TCP port used by IMEX for peer communication is defined by SERVER_PORT in /etc/nvidia-imex/config.cfg.

For running IMEX per job, the port in the file is written in as a random port for that job, in the allowed range of 0 to 65535.

To use an IMEX that is not managed by BCM, the SERVER_PORT value needs to be changed manually for all nodes to suit that other IMEX. This can be done with the help of the pdcp command, which comes with the pdsh package (section 14.1). The pdcp command can copy the config.cfg file from the BCM software image to all nodes that no longer use BCM to run IMEX per job.

Workload Power Profile Settings

NVIDIA Blackwell GPUs support workload power profile settings (WPPS). The *NVIDIA Mission Control System Administration Guide* has more background information on WPPS at:

https://docs.nvidia.com/mission-control/docs/systems-administration-guide/2.0.0/wpps/introduction.html

For WPPS to work as expected, a node that has a job allocated to it must have the job allocated exclusively to it, and oversubscribe (page 203) should not be set.

Each GPU in a job can have a specific workload power profile set for it. The profile can vary per GPU and per user. The gpuworkloadpowerprofiles command (page 396) can be used to configure the GPU workload power profiles.

BCM allows the user who is running a Slurm job to change the job power profiles in two ways:

1. Statically: the job prolog script switches the power profiles for all GPUs allocated to the job just before the job starts. That is, the power profile is managed at job submission time.
2. Dynamically: the job process sets power profiles for each allocated GPU during job execution. That is, the power profile is managed during the job runtime.

In both cases, when the job is finished, the epilog script resets the GPU power profiles back to the default.

Workload power profiles settings are a part of the slurmclient role, located within the Power profiles submode. They can be reached from within the configuration overlay, category, or node level:

Example

```
[root@basecm11 ~]# cmsh
[basecm11]% configurationoverlay roles slurm-client-gpu
[basecm11->configurationoverlay[slurm-client-gpu]->roles]% use slurmclient
[basecm11->configurationoverlay[slurm-client-gpu]->roles[slurmclient]]% powerprofiles
[basecm11->configurationoverlay[slurm-client-gpu]->roles[slurmclient]->powerprofiles]% show
Parameter                      Value
------------------------------ ------------------------------------------------
Disabled                       yes
Fail on error                  no
Job keyword                    wpps
Jobs profiles directory        /var/run/nvidia/workload-power-profiles
Debug                          no
Debug log directory            /var/spool/cmd/wlm/wpps
[basecm11->configurationoverlay[slurm-client-gpu]->roles[slurmclient]->powerprofiles]% set disabled no
```

```
[basecm11->configurationoverlay*[slurm-client-gpu*]->roles*[slurmclient*]->powerprofiles*]% commit
[basecm11->configurationoverlay[slurm-client-gpu]->roles[slurmclient]]%
```

With Base View, the equivalent navigation path to the preceding is:

`Grouping > Configuration Overlays > slurm-client-gpu` > `Settings > Roles` > `slurmclient` > `Power profiles`

The `Power profiles` parameters as shown in `cmsh` are:

1. `Disabled`: forbids a job from changing workload power profiles (`yes` by default)
2. `Fail on error`: allows prolog and epilog to fail if a workload power profiles change fails for any reason
3. `Job keyword`: a keyword, `wpps`, in the JSON object, used by the WLM job, to specify a workload power profiles setting in the comment field.
4. `Jobs profiles directory`: a top directory which includes job ID directories, and used by the job to change the workload power profiles
5. `Debug`: setting it to `yes` enables prolog and epilog debug messages The prolog and epilog then write debug logs to the location set by the `Debug log directory` parameter,
6. `Debug log directory`: directory where prolog and epilog create debug log files per job. The logs take the format: `/var/spool/cmd/wlm/wpps/<`*JOB ID*`>.log` by default. The log files are not cleaned up automatically.

If WPPS is enabled then:

- If a job requires static job profiles management, then special BCM prolog and epilog scripts are enabled in Slurm on compute nodes. The scripts parse the `SLURM_JOB_COMMENT` environment variable that is passed by Slurm. The scripts search for a JSON object that includes the key `wpps`. The key used can be changed using the `Job keyword` parameter. The expected format of the JSON object is:

  ```
  {"wpps": {"profiles": PROFILES}}
  ```

  Here, `PROFILES` is either

  - a JSON list of strings (workload profile names or their numbers) or
  - a single string (one workload profile name) or
  - a workload profile number.

  The values that `PROFILES` can take are:

  - `max_p`
  - `max_q`
  - `llm_training`
  - `llm_inference`

  On matching, the specified profiles are set for all the GPUs on the node that are allocated for the job when the job starts.

  For example, the following job submission command instructs the prolog script to set the workload profiles to `max_p` and `compute`:

```
$ sbatch --gres=gpu:b100:8 --comment='{"wpps": {"profiles": ["max_p", "compute"]}}' job.sh
```

All allocated GPUs on the node get the specified workload profiles, which means that job allocation must be configured to run on that node as a job that excludes other jobs running on it. If the cluster administrator has set oversubscribe (page 203) for Slurm, then that takes precedence over exclusivity, and must therefore be taken into account.

- If a job requires dynamic profiles management, then the job process can notify BCM to update the power profiles of specific GPUs during job execution. The prolog creates a directory of the form `/var/run/nvidia/workload-power-profiles/<JOB_ID>/` per job. The following files are created within the directory:

  1. `username`: read-only (for users) file that includes the current job user name. Used by BCM.
  2. `uuids`: read-only (for users) file that includes a list of GPU UUIDs (one per line) allocated for this job.
  3. `profiles`: file that can be used by the job process to set new profiles. The file is empty by default. An empty file means that BCM does not modify the profiles. If the job process writes two lines:

     ```
     max_p
     compute
     ```

     then BCM reads this file almost immediately, and applies these two workload profiles to the GPUs whose UUIDs are specified in `uuids` file.

What users are running in each workload power profile can be viewed with a PromQL query, as explained on page 400.

Management of node workload profiles using `gpuworkloadpowerprofiles`, by regular users: The user, for example `John`, can be given access to checking and managing GPU power profiles by running the `allowgpuworkloadpowerprofiles` command.

Example

```
[basecm11->user[john]]% set allowgpuworkloadpowerprofiles yes
```

The `gpuworkloadpowerprofiles` command can then be run to check the current node profiles:

Example

```
[basecm11->device]% gpuworkloadpowerprofiles show -n node001
Node         GPU      Profiles
------------ -------- ------------
node001      0        COMPUTE(2)
node001      1        MAX_P(0)
node001      2        MAX_P(0)
node001      3        MAX_P(0)
node001      4        MAX_P(0)
node001      5        MAX_P(0)
node001      6        MAX_P(0)
node001      7        MAX_P(0)
[basecm11->device]%
```

This may be useful for debug purposes.

The BCM script `cm-gpu-workload-power-profiles` can then be used to change the profiles as follows:

- The user first gets the GPU UUIDs:

  Example

  ```
  root@dgx-gb200-n07-c2:~# nvidia-smi --query-gpu=uuid --format=csv,noheader --id=0,1,2,3
  GPU-1e72bc8d-b967-6031-24bd-c5a08ad090e1
  GPU-3a6ac832-3423-68cc-ed1f-64f2b46d248d
  GPU-f40fabb7-eb9b-0253-4bd5-08b4a12916ea
  ```

- The user can then use the script to change the profile. For example, just for the GPUs 1 and 2:

  Example

  ```
  root@dgx-gb200-n07-c2:~# /cm/local/apps/cmd/sbin/cm-gpu-workload-power-profiles -p compute -u john \
  3a6ac832-3423-68cc-ed1f-64f2b46d248d f40fabb7-eb9b-0253-4bd5-08b4a12916ea
  ```

  The script carries the profile change out with the help of CMDaemon in the back end.

  Slurm's prolog and epilog actually work with the script in BCM when carrying out dynamic profiles management, so that the user does not need to use the script manually.

Management of external users to allow them to use profiles: External users, that is users managed by an external LDAP and not managed by the BCM LDAP, can use the workload power profile settings if their user name or UID is added to the file:

```
/cm/local/apps/cmd/etc/allow-users-wpps.conf
```

The file must be created on the head node, or on both head nodes if the cluster has HA. The file should have its mode set to 0600. The user name or UID should be added on its own line in the file.

Advanced Slurm Job Accounting

PromQL queries can be run and then filtered by Slurm job labels with BCM. The labels are taken either from a job comment or from an account name used by the job. After the labels for the job are extracted, the labeled entity is stored in the monitoring data. The administrator can then run PromQL queries, and filter results by label. Examples of such PromQL queries are the `job_gpu_wasted` or `job_gpu_utilization` queries (section 12.4.1).

`Extract accounting info` must be set to `yes` to enable labels to be extracted from job comments or account names:

Example

```
[root@basecm11 ~]# cmsh
[basecm11]% wlm use slurm
[basecm11->wlm[slurm]]% accounting
[basecm11->wlm[slurm]->show]% accounting
Parameter                      Value
------------------------------ ------------------------------------------------
Managed hierarchy
Separator                      _
Job comment labels
Extract accounting info        no
[basecm11->wlm[slurm]->accounting]% set extractaccountinginfo yes
[basecm11->wlm*[slurm*]->accounting*]% commit
```

In addition, the administrator must configure the job comment label format and the format of Slurm account names, if they exist. Without that configuration, BCM cannot parse the labels.

Job comment labels: A user running the job can specify the label for the job comment in a special format. Such a label could be, for example, a tag to indicate the artificial neural network model, where different groups of jobs use different models. The administrator (after having enabled accounting information extraction) can set a regex for the tag in a `Job comment field` parameter in `accounting` submode:

Example

```
[root@basecm11 ~]# cmsh
[basecm11]% wlm use slurm
[basecm11->wlm[slurm]]% accounting
[basecm11->wlm[slurm]->accounting]% set jobcommentlabels model
[basecm11->wlm*[slurm*]->accounting*]% commit
```

In the preceding example, the label `model` is set in the monitoring data when the job comment includes a JSON object with `model` as a key. The value must be a string that will be a value for the label `model`. For example `"model": "llm123"`. The JSON object may contain other information that will be ignored by BCM when it reads the labels from the comment.

Account name labels: Several labels can be assigned by BCM to a job parsed from an associated account name. The labels can be used to represent a hierarchy of organizational entities, such as department, project, team, and so on. The following configuration options allow this representation:

1. `Managed hierarchy`: representation of the account name as a list of organizational entity labels.
2. `Separator`: a separator for the labels in the account names. By default this is the underscore character, _. The last entity is separated with double separator. This means that the last entity (and only the last entity) can have a single separator in the name itself.

For instance, if Slurm account names use the following format:

```
DEPARTMENT_PROJECT_TEAM
```

then `Managed hierarchy` values can be set as follows:

Example

```
[root@basecm11 ~]# cmsh
[basecm11]% wlm use slurm
[basecm11->wlm[slurm]]% accounting
[basecm11->wlm[slurm]->accounting]% set managedhierarchy department project team
[basecm11->wlm*[slurm*]->accounting*]% commit
```

It is up to the administrator to create the accounting hierarchy in Slurm, and set names as described by the `Managed hierarchy` parameter.

Example

```
[root@basecm11 ~]# module load slurm
[root@basecm11 ~]# sacctmgr add account NVIDIA Description="NVIDIA account" Organization=NVIDIA parent=root -i
[root@basecm11 ~]# sacctmgr add account Department1 Organization=NVIDIA parent=NVIDIA -i
[root@basecm11 ~]# sacctmgr add account Department1_Project1 Organization=NVIDIA parent=Department1 -i
[root@basecm11 ~]# sacctmgr add account Department1_Project1__Team_1 Organization=NVIDIA parent=Department1 -i
[root@basecm11 ~]# sacctmgr add account Department1__Team_2 Organization=NVIDIA parent=Department1 -i
```

In the preceding example two teams have been made. Team 1 is defined according to the full entities hierarchy (department, project, team), while Team 2 is defined only under a department. BCM correctly recognizes such "clipped" hierarchy locations for a team in the account names.

Similar to the Slurm command `sshare`, an administrator can display the Slurm account hierarchy with the `fairshare` command in `cmsh`:

Example

```
[basecm11->wlm[slurm]]% fairshare -t
Account                        fairshare    raw_shares raw_usage  norm_shares norm_usage parent
------------------------------ ------------ ---------- ---------- ----------- ---------- ------------------
root (top account)             0.0          0          0          0.0         0.0
nvidia                         0.0          1          0          0.5         0.0        root (top account)
department1                    0.0          1          0          1.0         0.0        nvidia
department1__team_2            0.0          1          0          0.333333    0.0        department1
department1_project1           0.0          1          0          0.333333    0.0        department1
department1_project1__team_1   0.0          1          0          0.333333    0.0        department1
root                           1.0          1          0          0.5         0.0        root (top account)
[basecm11->wlm[slurm]]% fairshare -f ""
root
├─nvidia
│  └─department1
│     ├─department1__team_2
│     ├─department1_project1
│     └─department1_project1__team_1
└─root
```

The following command options can be used:

1. `-t|--table`: print accounts fairshare parameters as a table
2. `-a|--account <name>`: print information for specific account
3. `-f|--fields <PARAMETERS>`: display only specified fairshare account parameters (comma-delimited) Format:

   ```
   <FIELD>[%<MIN>[-<MAX>]][,<FIELD>[%<MIN>[-<MAX>]],...] ;
   ```

   where `MIN` and `MAX` are minimum and maximum value lengths. The parameter can be combined with others.

The following example uses labels from both job comment and account names:

Example

```
[root@basecm11 ~]# su - alice
[alice@basecm11 ~]$ module load slurm
[alice@basecm11 ~]$ sbatch -A department1_project1__team_1 --comment='{"model": "llm123"}' --wrap="hostname"
Submitted batch job 1
[alice@basecm11 ~]$
```

PromQL queries with these labels are run later on.

In `cmsh` the administrator can validate how BCM parses the job labels from both job comment and from the Slurm account name:

```
[basecm11->wlm[slurm]->jobs]% info 1 | grep "accounting info" -i
Accounting info       {"department":"department1","model":"llm123","project":"project1","team":"team_1"}
[basecm11->wlm[slurm]->jobs]%
```

Jobs information can also be displayed by filtering using the labels (some output elided for clarity):

```
[basecm11->wlm[slurm]->jobs]% filter -i team="team_1"
Job ID  Job name   User   Queue   Submit time  Start time  End time  Nodes     Exit code
------- ---------- ------ ------- ------------ ----------- --------- --------- ----------
1       wrap       alice          13:28:46     13:28:46    13:28:46              1
2       wrap       alice  defq    13:28:59     13:28:59    13:30:00  node001   0
3       wrap       alice  defq    13:30:39     13:30:39    13:31:40  node001   0
[basecm11->wlm[slurm]->jobs]%
```

The following PromQL query (section 12.3) filters the team:

```
[basecm11->monitoring->labeledentity]% instantquery "job_gpu_wasted{team=\"team_1\"}[1h]"
Name             category department   group  hostname job_id  job_name project   queue  team    user   wlm   ...
---------------- -------- ------------ ------ -------- ------- -------- --------- ------ ------- ------ ----- ...
job_gpu_wasted   default  department1  alice  node001  1       wrap     project1  defq   team_1  alice  slurm ...
job_gpu_wasted   default  department1  alice  node002  2       wrap     project1  defq   team_1  alice  slurm ...
```

The following query filters the model name:

```
[basecm11->monitoring->labeledentity]% instantquery "job_gpu_wasted{model=\"llm123\"}[1h]"
Name             category group  hostname job_id   job_name model    queue    user   wlm   ...
---------------- -------- ------ -------- -------- -------- -------- -------- ------ ----- ...
job_gpu_wasted   default  alice  node001  1        wrap     foo      defq     alice  slurm ...
job_gpu_wasted   default  alice  node002  2        wrap     foo      defq     alice  slurm ...
```

The following query shows what users are running in each WPPS profile:

```
[basecm11->monitoring->labeledentity]% instantquery -q job_gpu_workload_power_profile_for_user
Name                                    gpu_workload_power_profile user       Timestamp  ... Value
--------------------------------------- -------------------------- ---------- -----------... -----
job_gpu_workload_power_profile_for_user  COMPUTE                    avolkov    Mon Jun  2 ... 2
job_gpu_workload_power_profile_for_user  LLM_INFERENCE              avolkov    Mon Jun  2 ... 2133
job_gpu_workload_power_profile_for_user  LLM_INFERENCE              root       Mon Jun  2 ... 23
job_gpu_workload_power_profile_for_user  LLM_INFERENCE              shoreline  Mon Jun  2 ... 336
job_gpu_workload_power_profile_for_user  LLM_INFERENCE              wglantz    Mon Jun  2 ... 12
job_gpu_workload_power_profile_for_user  LLM_TRAINING               root       Mon Jun  2 ... 7
job_gpu_workload_power_profile_for_user  LLM_TRAINING               shoreline  Mon Jun  2 ... 331
job_gpu_workload_power_profile_for_user  MAX_P                      avolkov    Mon Jun  2 ... 2134
job_gpu_workload_power_profile_for_user  MAX_P                      root       Mon Jun  2 ... 60
job_gpu_workload_power_profile_for_user  MAX_P                      shoreline  Mon Jun  2 ... 379
job_gpu_workload_power_profile_for_user  MAX_P                      wglantz    Mon Jun  2 ... 12
job_gpu_workload_power_profile_for_user  MAX_Q                      root       Mon Jun  2 ... 8
job_gpu_workload_power_profile_for_user  MAX_Q                      shoreline  Mon Jun  2 ... 320
```

Slurm PRS Settings

Power Reservation Steering (PRS) and its deployment is covered in Chapter 10 of the *NVIDIA Mission Control Manual*.

The `prssettings` submode in `wlm` mode configures the certificate paths used by the Slurm PRS plugin.

Example

```
[basecm11->wlm[slurm]->prssettings]% show
Certificate path                /cm/local/apps/prs/etc/${WLM_NAME}.pem
Private key path                /cm/local/apps/prs/etc/${WLM_NAME}.key
```

The `Certificate path` and `Private key path` settings specify the TLS certificate and key used for secure communication with the PRS service. The `${WLM_NAME}` variable is automatically substituted with the name of the WLM instance.

Configuring Slurm

After Slurm setup is configured and installed with `cm-wlm-setup` (section 7.3), the Slurm software components are installed in a symlinked directory `/cm/local/apps/slurm/current`. The same set of Slurm packages should be installed on the head nodes and in each software image where any of the Slurm services run.

Slurm clients and servers can be configured to some extent using role assignment (sections 7.4.1 and 7.4.2).

Using `cmsh`, advanced option parameters can be set under the `slurmclient` and `slurmserver` roles. The settings for the roles can be done at configuration overlay, category, or node level (sections 2.1.5, 2.1.6).

By default, the `cm-wlm-setup` utility configures Slurm using configuration overlays.

Example

```
[basecm11->configurationoverlay]% list
Name (key)           Priority   All head nodes Nodes            Categories       Roles
-------------------- ---------- -------------- ---------------- ---------------- ----------------
slurm-accounting     500        yes                                              slurmaccounting
slurm-client         500        no                               default          slurmclient
slurm-server         500        yes                                              slurmserver
slurm-submit         500        no                               default          slurmsubmit
wlm-headnode-submit  600        yes                                              slurmsubmit
```

The settings within the roles can be viewed and modified. For example, the `slurmclient` role of the `slurm-client` configuration overlay can be viewed:

Example

```
[basecm11->configurationoverlay]% roles slurm-client
[basecm11->configurationoverlay[slurm-client]->roles]% show slurmclient
Parameter                      Value
------------------------------ ------------------------------------------------
Name                           slurmclient
Revision
Type                           SlurmClientRole
Add services                   yes
WLM cluster                    slurm
Slots                          0
All Queues                     no
Queues                         defq
Features
Sockets                        0
Cores Per Socket               0
ThreadsPerCore                 0
Boards                         0
SocketsPerBoard                0
RealMemory                     0B
NodeAddr
Weight                         0
Port                           0
TmpDisk                        0
Reason
CPU Spec List
Core Spec Count                0
Mem Spec Limit                 0B
```

```
GPU auto detect                  BCM
Node Customizations              <0 in submode>
Generic Resources                <0 in submode>
Cpu Bindings                     None
Slurm hardware probe autodetect  yes
Memory autodetection slack       0.0%
IMEX                             no
```

Assigning nodes to Slurm queues: Slurm can configure *nodesets* (`man slurm.conf.5`). Nodesets are a way to conveniently group nodes under a unique arbitary name, so that features can be assigned to a group of nodes.

Starting with BCM version 11, BCM can be used to configure Slurm nodesets using the `Nodesets` and `Nodeset features` options within the `slurmclient` role options:

- `Nodesets`: An arbitrary name can be set for the `Nodesets` parameter. Names that are set are automatically added to `slurm.conf`. Nodes with the `slurmclient` role are then associated with this arbitrary nodeset name in `slurm.conf`. For example, if the `nodesets` parameter is set to `ns1` and the role is assigned only to node `node001`, then BCM translates this configuration after a short time into a `slurm.conf` line that looks like:

  `NodeSet=ns1 Nodes=node001`

  In an express Slurm setup by `cm-wlm-setup` (page 361), the `slurmclient` role is set by default for all nodes using the configuration overlay. This means that setting the `nodesets` parameter in the `slurmclient` role there sets it for all nodes. After a short while, the `NodeSet` line in `slurm.conf` changes to match the parameter change:

  Example

  ```
  [basecm11->configurationoverlay[slurm-client]->roles[slurmclient] get nodesets
  [basecm11->configurationoverlay[slurm-client]->roles[slurmclient] set nodesets ns1; commit
   then, after a minute or so:
  [basecm11->...->roles[slurmclient] !grep -i nodeset /cm/shared/apps/slurm/etc/slurm/slurm.conf
  [basecm11->...->roles[slurmclient] NodeSet=ns1 Nodes=node[001-003]      for a 3-node cluster
  ```

- `Nodeset features`: All values of the `Nodeset features` parameter are added to nodes that have the `slurmclient` role. They are added as Slurm node features, which means that there is no need to duplicate them in the `Features` parameter of the `slurmclient` role. The values that are set are a list of strings.

  The Slurm nodesets are added to `slurm.conf` with the same name as these features. This is useful when the administrator needs to assign sets of nodes to queues based on the node features.

  For example, if the parameter has a value `bigmem`, and the `slurmclient` role is assigned to node `node001`, then BCM adds lines with:

  `NodeName=node001 Features=bigmem ...`
  and
  `NodeSet=bigmem Feature=bigmem`

  to `slurm.conf`.

  For the 3-node case specified by the configuration overlay of earlier, the change is illustrated by the following:

  Example

```
[basecm11->...->roles[slurmclient]]% get nodesetfeatures
[basecm11->...]->roles[slurmclient]]% !grep -i ^Node /cm/shared/apps/slurm/etc/slurm/slurm.conf
NodeName=node[001-003] ... Features=location=local
NodeSet=ns1 Nodes=node[001..003]
[basecm11->...->roles[slurmclient]]% set nodesetfeatures bigmem
... wait a little for the change to happen...
[basecm11->...->roles[slurmclient]]% !grep -i ^Node /cm/shared/apps/slurm/etc/slurm/slurm.conf
NodeName=node[001-003] ... Features=location=local,bigmem
NodeSet=ns1 Nodes=node[001-003]
NodeSet=bigmem Feature=bigmem
```

The `Nodeset features` parameter does not automatically add the nodesets to any queues. The administrator must add the nodesets to the queues separately.

Specifying nodesets for Slurm using `wlm` mode with `jobqueues`, vs specifying nodes using the `slurmclient` role's `queues` parameter:

Specifying which nodes go into Slurm job queues can be specified using either nodesets or nodes.

- The administrator can specify which nodesets are included in Slurm job queues by setting the `nodesets` parameter for a job queue, within a `jobqueue` submode.

  For example, the nodeset `ns1` defined in the `slurmclient` role earlier can be included in `defq` with:

  Example

  ```
  root@basecm11:~# cmsh
  [basecm11]% wlm jobqueue
  [basecm11->wlm[slurm]->jobqueue]% use defq
  [basecm11->wlm[slurm]->jobqueue[defq]]% set nodesets ns1; commit
  ```

  If the `nodesets` parameter is set for a queue, then BCM does not add any nodes that are specified by the `Compute nodes` parameter, but only adds the nodes in the `nodesets` specification to a line with the format:

  `PartitionName=...  Nodes=`<*nodesets*>` ...`

  in `slurm.conf`. As a reminder, a queue in Slurm terminology is a partition.

- The administrator can specify the nodes that are associated with a job queue.

  – Prior to BCM version 11, job queues were specified for nodes only by using the `queues` parameter of the `slurmclient` role at the `device`, `category`, or `configurationoverlay` mode level.

    For example, for the configuration overlay `slurm-client` that defines the Slurm client configuration by default, the nodes for the Slurm client role are the nodes in the default category `default`. Within the configuration overlay, within the `slurmclient` role, the default queue to be used is set to `defq` by default:

    Example

    ```
    root@basecm11:~# cmsh
    [basecm11]% configurationoverlay use slurm-client
    [basecm11->configurationoverlay[slurm-client]]% get categories
    default
    [basecm11->configurationoverlay[slurm-client]]% roles
    [basecm11->configurationoverlay[slurm-client]->roles]% use slurmclient
    [basecm11->configurationoverlay[slurm-client]->roles[slurmclient]]% get queues
    defq
    ```

– Starting with BCM version 11, the administrator can also add nodes to queues directly from within `wlm` mode for a particular WLM, within the `jobqueue` mode for a particular queue, by setting nodes for particular node grouping parameters. For example, for Slurm the following queue parameters are available:

Example

```
[basecm11->wlm[slurm]->jobqueue[defq]]% show | egrep -i '(overlay|nodegroups|compute|catego|nodeset)'
Nodesets
Overlays
Categories
Nodegroups
Compute nodes
```

Thus `defq` is used by nodes that have been specified in the following groupings:

* `Nodesets`: nodes in these nodesets
* `Overlays`: nodes in these configuration overlays (including nodes of categories that are in this overlay)
* `Categories`: nodes in these categories
* `Nodegroups`: nodes in these node groups
* `Compute nodes`: nodes listed in this parameter

Slurm Hardware Autodetection (non-GPU): The setting `Slurm hardware probe autodetect` in the Slurm role enables automated hardware detection for Slurm for non-GPU hardware. (Specifically for GPUs is the related Slurm `GPU auto detect` setting (page 405)).

`Slurm hardware probe autodetect` enables detection for the following Slurm parameters:

- `corespersocket`
- `threadspercore`
- `boards`
- `socketsperboard`
- `sockets`
- `realmemory`

The autodetected parameters are placed in the `NodeName` line in `slurm.conf`.

The Slurm client role parameters can be modified. For example `Core Spec Count`:

Example

```
[basecm11->configurationoverlay[slurm-client]->roles]% set slurmclient corespeccount 2
[basecm11->configurationoverlay*[slurm-client*]->roles*]% commit
```

As usual, values set at node level override the values set at categories and configuration overlays level.

For example, to set `corespeccount` to 4, only for node001 but not for other nodes, the session might run further as:

Example

```
[basecm11->configurationoverlay[slurm-client]->roles]% device use node001
[basecm11->device[node001]]% roles
[basecm11->device[node001]->roles]% assign slurmclient
[basecm11->device*[node001*]->roles*[slurmclient*]]% set corespeccount 4
[basecm11->device*[node001*]->roles*[slurmclient*]]% commit
Field                    Message
------------------------ ------------------------------------------------------------
wlmCluster               Error: The WLM cluster should be set
[basecm11->device*[node001*]->roles*[slurmclient*]]% wlm list
Type   Name (key)               Server nodes Submit nodes     Client nodes
------ ------------------------ ------------ ---------------- ----------------
Slurm  slurm                    basecm11     basecm11,node001 node001,node001
[basecm11->device*[node001*]->roles*[slurmclient*]]% set wlmcluster slurm
[basecm11->device*[node001*]->roles*[slurmclient*]]% commit
```

In the preceding session, the role needs to be assigned at node level with `assign slurmclient` because it does not initially exist at node level. If it already existed, then `use slurmclient` could have been used to descend into that role.

Also in the preceding session, one of the values that the Slurm client needs to know is `wlmcluster`, which decides which WLM it is to work with on the cluster. The value is selected from the list of WLM instance names in `wlm` mode.

The level of the active role can be seen with the `list` command. For example, the Slurm client role assignment at node level is seen here:

Example

```
[basecm11->device[node001]->roles[slurmclient]]% list
Name (key)
----------------------------------
[overlay:slurm-submit] slurmsubmit
slurmclient
```

Removing the assignment has the `list` command display the configuration overlay Slurm client role assignment:

Example

```
[basecm11->device[node001]->roles[slurmclient]]% unassign slurmclient; commit
[basecm11->device[node001]->roles[slurmclient]]% list
Name (key)
----------------------------------
[overlay:slurm-submit] slurmsubmit
[overlay:slurm-client] slurmclient
```

Slurm Hardware Autodetection For GPUs: The Slurm `GPU auto detect` setting is a setting seen in the session output of page 401. It manages Slurm GRES configuration (`https://slurm.schedmd.com/gres.conf.html`) automatically, and can be set:

- globally, for all Slurm compute nodes in a Slurm instance, from within the `wlm` mode of `cmsh`
- for a particular role, such as in a device role, a category role, and a configuration overlay role for a Slurm client

  Example

  ```
  [basecm11->configurationoverlay[slurm-client]->roles[slurmclient]]% set gpuautodetect <TAB><TAB>
  bcm     none    nrt     nvml    off     oneapi  rsmi
  ```

If GPU auto detect is assigned a value, then a corresponding value is assigned to the autodetect parameter in Slurm's gres.conf file. One out of 7 values can be assigned to GPU auto detect:

1. bcm: to use BCM values, which are the values that CMDaemon automatically puts in for the vendor (NVIDIA, Intel, AWS, AMD) and for the number of GPUs.
2. none: to not have the autodetect parameter exist in gres.conf. In this case, there is actually no corresponding value that can be assigned to the autodetect of gres.conf.
3. nrt: to detect AWS Trainium/Inferentia devices. <— not yet, in October 2024, trunk, or bcm10.24.09
4. nvml: to detect NVIDIA GPUs.
5. off: to turn Slurm GPU autodetection off.
6. oneapi: to detect Intel GPUs.
7. rsmi: to detect AMD GPUs

Slurm GPU autodetection is described further on page 425.

Slurm accounting database configuration in cmsh: After package setup is carried out with cm-wlm-setup (section 7.3), cmsh can be used to modify the settings for the Slurm accounting database. Changes can be carried out in the slurm-accounting configuration overlay:

```
[basecm11->configurationoverlay[slurm-accounting]]% show
Parameter                      Value
------------------------------ --------------------------
Name                           slurm-accounting
Revision
All head nodes                 no
Priority                       500
Nodes                          node001,node002
Categories
Roles                          slurmaccounting
Customizations                 <0 in submode>
```

In the preceding overlay, slurmdbd runs on the accounting node(s). It is possible to run slurmdbd on both head nodes or on one or two compute nodes, as set in the configuration overlay. It is not possible to mix up head nodes and compute nodes for this configuration, and it is not possible to run slurmdbd on more than two nodes.

Further Slurm accounting changes can be carried out within the slurmaccounting role:

```
[basecm11->configurationoverlay[slurm-accounting]->roles[slurmaccounting]]% show
Parameter                      Value
------------------------------ -------------------------
Name                           slurmaccounting
Revision
Type                           SlurmAccountingRole
Add services                   yes
High availability              yes
Primary accounting server      node001
DbdPort                        6819
StorageHost                    node001
StoragePort                    3306
StorageLoc                     slurm_acct_db
StorageUser                    slurm
```

The primaryaccountingserver parameter defines which node is primary, that is, which one is AccountingStorageHost in slurm.conf. The other node setting in the configuration overlay is the AccountingStorageBackupHost.

StorageHost, StoragePort, StorageLoc and StorageUser are settings for connecting to the MySQL database, If these are for an external host, then they must be configured manually so that they are reachable by the nodes running slurmdbd.

The highavailability parameter sets the high availability mode for slurmdbd. If set to yes, then the service runs on both nodes at the same time.

Generic resources (gres) configuration in Slurm: In order to configure generic resources, the genericresources mode can be used to set a list of objects. Each object then represents one generic resource available on nodes.

Each value of name in genericresources must already be defined in the list of GresTypes. The list of GresTypes is defined in the wlm role for the instance.

Example

```
[basecm11->wlm[slurm]]% get grestypes
gpu
```

Several generic resources entries can have the same value for name (for example gpu), but must have a unique alias. The alias is a string that is used to manage the resource entry in cmsh or in Base View. The string is enclosed in square brackets in cmsh, and is used instead of the name for the object. The alias does not affect Slurm configuration.

For example, to add two GPUs for all the nodes in the default category which are of type k20xm, and to assign them to different CPU cores, the following cmsh commands can be run:

Example

```
[basecm11]% configurationoverlay use slurm-client
[basecm11->configurationoverlay[slurm-client]]% roles
[basecm11->configurationoverlay[slurm-client]->roles*]% use slurmclient
[...[slurmclient]]% genericresources
[...[slurmclient]->genericresources]% add gpu0
[...[slurmclient*]->genericresources*[gpu0*]]% set name gpu
[...[slurmclient*]->genericresources*[gpu0*]]% set file /dev/nvidia0
[...[slurmclient*]->genericresources*[gpu0*]]% set cores 0-7
[...[slurmclient*]->genericresources*[gpu0*]]% set type k20xm
[...[slurmclient*]->genericresources*[gpu0*]]% add gpu1
[...[slurmclient*]->genericresources*[gpu1*]]% set name gpu
[...[slurmclient*]->genericresources*[gpu1*]]% set file /dev/nvidia1
[...[slurmclient*]->genericresources*[gpu1*]]% set cores 8-15
[...[slurmclient*]->genericresources*[gpu1*]]% set type k20xm
[...[slurmclient*]->genericresources*[gpu1*]]% commit
[...[slurmclient]->genericresources[gpu1]]% list
Alias (key) Name     Type     Count    File
----------- -------- -------- -------- ----------------
gpu0        gpu      k20xm             /dev/nvidia0
gpu1        gpu      k20xm             /dev/nvidia1
[...[slurmclient]->genericresources[gpu1]]%
```

Typically this configuration is done automatically during the GPU configuration process as outlined in the GPU Configuration Screens section on page 361, where a configuration overlay for the GPUs is given the default name slurm-client-gpu.

In Base View, the navigation path:

Grouping > Configuration Overlays > slurm-client-gpu > Settings > Roles > slurmclient > Generic resources > + Add

provides the equivalent (figure 7.22):

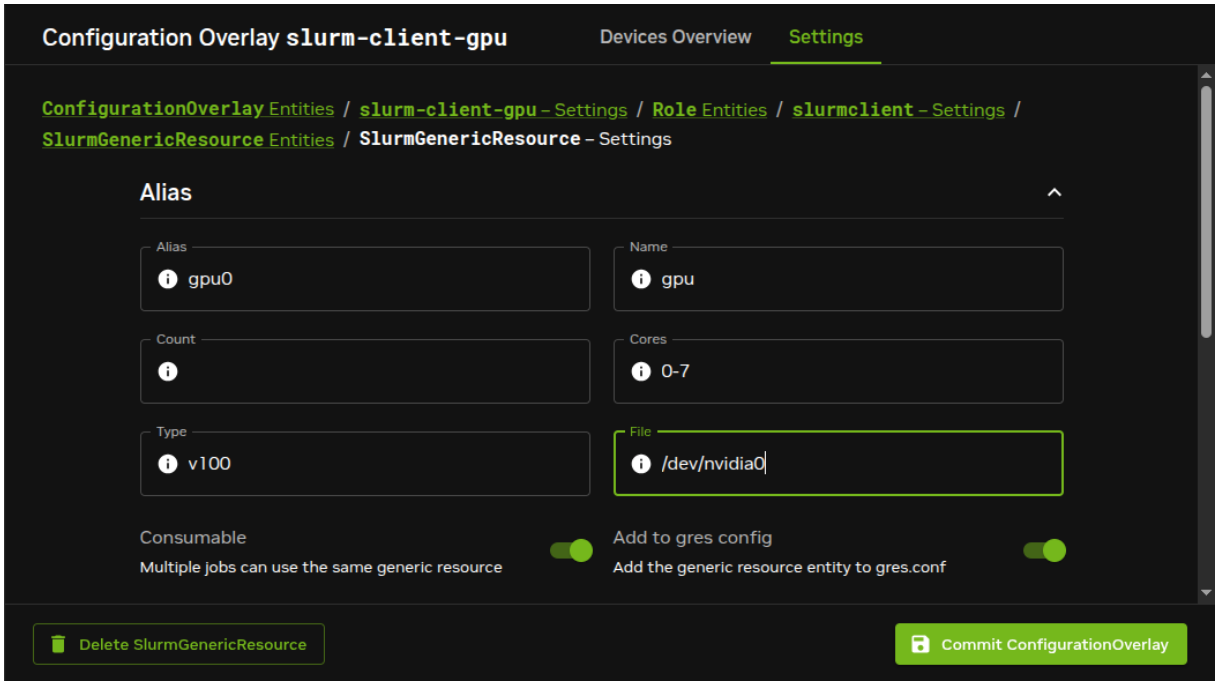


Figure 7.22: Base View access to NVIDIA GPU configuration options

After the generic resources are committed, BCM updates the `gres.conf` file.

Since NVIDIA Base Command Manager version 8.2, a single `gres.conf` configuration file, located at `/cm/shared/apps/slurm/etc/slurm/gres.conf` is used.

If the category consists of `node001` and `node002`, then the entries to the `gres.conf` file in this case would look like:

Example

```
# This section of this file was automatically generated by cmd. Do not edit manually!
# BEGIN AUTOGENERATED SECTION -- DO NOT REMOVE
NodeName=node[001,002] Name=gpu Type=k20xm Count=1 File=/dev/nvidia0 Cores=0-7
NodeName=node[001,002] Name=gpu Type=k20xm Count=1 File=/dev/nvidia1 Cores=8-15
# END AUTOGENERATED SECTION   -- DO NOT REMOVE
[root@basecm11 ~]#
```

Slurm topology configuration: Slurm supports topology-aware resource allocation to optimize job performance. BCM supports configuration for the `topology/tree` and `topology/block` Slurm plugins using the `topograph` service (page 413):

- `topology/tree`: models the cluster network as a hierarchical structure, where Slurm allocates resources to jobs to minimize network contention. Nodes are grouped under leaf switches, which connect to higher-level switches. The plugin helps to allocate nodes that are close together in the network hierarchy. The plugin requires all nodes for a single job be connected using switches. The configuration of the plugin is defined using switches and their relationships.

- `topology/block`: models the network as non-overlapping blocks of nodes. The plugin aims to keep jobs contiguous within blocks for better communication. The plugin does not require all nodes to be part of blocks. The configuration is defined using blocks of nodes and their groupings.

The network topology configuration for a Slurm instance <*slurmcluster*> is described by:
`/cm/shared/apps/slurm/etc/<slurmcluster>/topology.conf`

The file is modified in an autogenerated section when the cluster administrator modifies the internal topology settings in BCM, or when the topology arrangement record is fetched from Topograph, the topology generation service (page 413). The topology settings can be modified in BCM via the `topologysettings` submode of the WLM instance in `cmsh`:

Example

```
[root@basecm11 ~]# cmsh
[basecm11]% wlm use slurm
[basecm11->wlm[slurm]]% topologysettings
[basecm11->wlm[slurm]->topologysettings]% show
Located Parameter                        Value
-------------------------------- ------------------------------------------------
Topology plugin                  Block
Topology source                  Internal
Parameters                       <submode>
Tree settings                    <submode>
Block settings                   <submode>
Topograph settings               <submode>
```

A visualization (figure 7.23) of `topologysettings`, and its submodes and settings may be useful when reading the contents of this section:

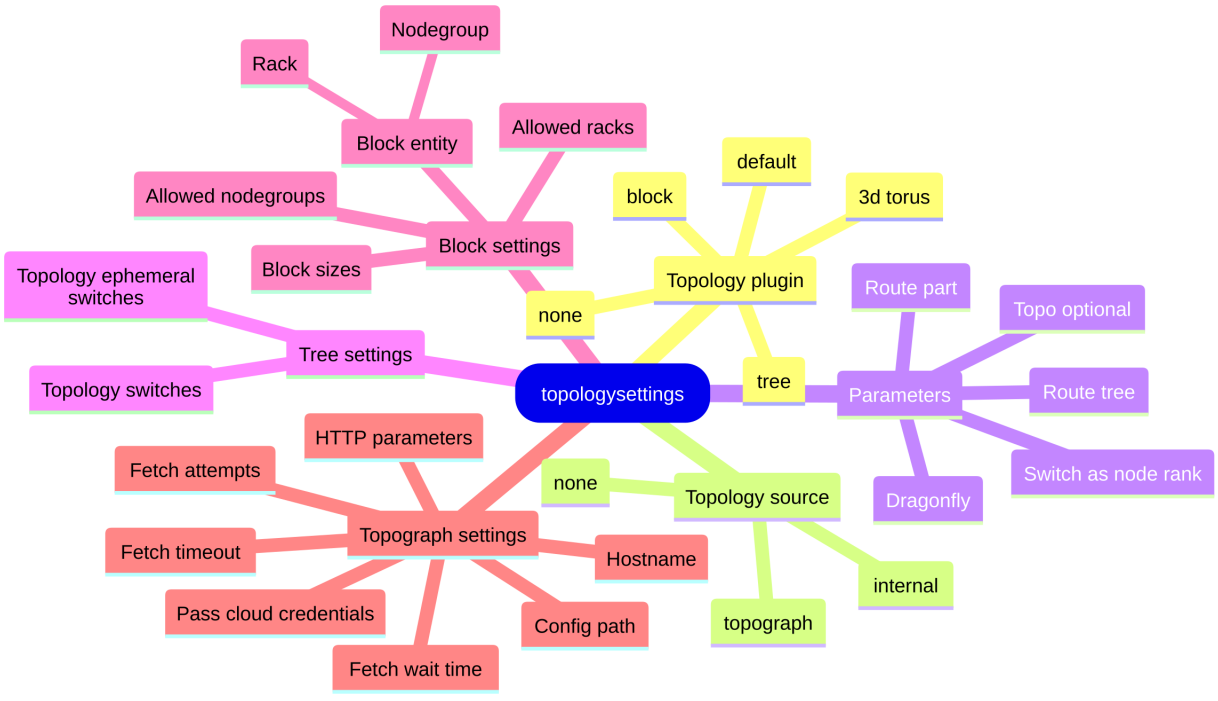


Figure 7.23: Visualization of settings under `topologysettings`

BCM allows a topology configuration to be generated for both the `topology/tree` and `topology/block` Slurm plugins. By default neither of these plugins is enabled. To enable topology

generation, the administrator can select the plugin type within WLM mode using the `Topology plugin` parameter, and can select the `Topology source`:

```
[root@basecm11 ~]# cmsh
[basecm11]% wlm use slurm
[basecm11->wlm[slurm]]% set topologyplugin<TAB><TAB>
3d torus   block   default   none   tree
[basecm11->wlm*[slurm*]]% set topologyplugin tree
Tree
[basecm11->wlm*[slurm*]]% set topologysource<TAB><TAB>
internal   none       topograph
[basecm11->wlm*[slurm*]]% set topologysource internal
[basecm11->wlm*[slurm*]]% commit
```

By default, both `Topology source` and `Topology plugin` are set to `none`, which means that `topology.conf` is not updated by BCM.

Within `topologysettings`, the `Topology plugin` parameter configures plugins in the Slurm-related `.conf` files:

Example

```
[basecm11->wlm[slurm]->topologysettings]% set topologyplugin<TAB><TAB>
3d torus  block      default    none       tree
```

- `none`: no plugin is configured in `slurm.conf`
- `default`: the `topology/default` plugin is set in `slurm.conf`
- `3d torus`: the `topology/3d_torus` plugin is set in `slurm.conf`
- `block`: the `topology/block` plugin is set in `slurm.conf` and `topology.conf` is updated
- `tree`: the `topology/tree` plugin is set in `slurm.conf` and `topology.conf` is updated

The topology settings can be configured in a similar way in Base View.

Within `topologysettings`, there are these 4 submodes: `Parameters`, `Tree settings`, `Block settings`, and `Topograph settings`:

1. The `parameters` submode manages parameters that are common for all plugins:

   Example

   ```
   [basecm11->wlm[slurm]->topologysettings]% parameters
   [basecm11->wlm[slurm]->topologysettings->parameters]% show
   Parameter                      Value
   ------------------------------ ------------------------------------------------
   Dragonfly                      no
   Route part                     no
   Switch as node rank            no
   Route tree                     no
   Topo Optional                  no
   ```

   These correspond to values of `TopologyParam` in `slurm.conf` (`man slurm.conf.5`).

2. The `treesettings` submode allows access to some parameters that modify the topology for the `topology/tree` plugin:

Example

```
[basecm11->wlm[slurm]->topologysettings]% treesettings
[basecm11->wlm[slurm]->topologysettings->treesettings]% show
Parameter                      Value
------------------------------ ------------------------------------------------
Topology switches
Topology ephemeral switches    yes
```

The `treesettings` parameters are:

- `topologyephemeralswitches`: A boolean to allow ephemeral switches to be added to `topology.conf`
- `topology switches`: A Slurm cluster configuration parameter that defines the list of switches used to write the topology file:

  Example

  ```
  [basecm11->wlm[slurm]->topologysettings->treesettings]% set topologyswitches switch01 switch02 switch03
  [basecm11->wlm[slurm*]]% commit
  ```

  The switches-to-nodes mapping does not require the switch devices to exist. The mapping can be defined in one of the following ways:

  – Definitions using the `switchports` parameter: the port definitions (page 146) for a node or a switch must normally be configured to let BCM construct the topology tree in `topology.conf`. The definitions can be carried out using their `switchports` parameter:

    Example

    ```
    [root@basecm11 ~]# cmsh
    [basecm11]% device use node001
    [basecm11->device[node001]]% set switchports switch01:3
    [basecm11->device[node001*]]% commit
    [basecm11->device[node001]]% use switch01
    [basecm11->device[switch01]]% set switchports switch02:15
    [basecm11->device[switch01*]]% commit
    ```

  – Definitions using the `SlurmTopology` parameter: the Slurm network topology can be configured without configuring the ports in BCM, by using their `SlurmTopology` parameter:

    Example

    ```
    [root@basecm11 ~]# cmsh
    [basecm11]% device
    [basecm11->device]% set node001 -e SlurmTopology switch02 switch01
    [basecm11->device*]% set node002 -e SlurmTopology switch02 switch01
    [basecm11->device*]% set node003 -e SlurmTopology switch03
    [basecm11->device*]% commit
    ```

    The preceding example populates the autogenerated section of `topology.conf` as follows:

    Example

    ```
    SwitchName=switch01 Switches=switch02
    SwitchName=switch02 Nodes=node[001,002]
    SwitchName=switch03 Nodes=node003
    ```

  The following `AdvancedConfig` directives (page 925) can be used to control the topology parameters layout further.

- SlurmStraightExtraTopology: If set to a value of 0, then the order of switches is reversed when setting the extra values. For example, to get the same content of topology.conf as in the preceding example, the following lines can then be committed:

  Example

  ```
  [root@basecm11 ~]# cmsh
  [basecm11]% device
  [basecm11->device]% set node001 -e SlurmTopology switch01 switch02
  [basecm11->device*]% set node002 -e SlurmTopology switch01 switch02
  [basecm11->device*]% set node003 -e SlurmTopology switch03
  [basecm11->device*]% commit
  ```

  In topology/plugin configuration, switches and nodes cannot both be connected to the same switch. This is consistent with the description in man topology.conf.5. If the configuration in BCM has both nodes and switches connected to the same switch, then only the nodes-to-switch connection is written to topology.conf.
- SlurmConcatTopologySwitchName: If set to 1, then it allows concatenation of the switch names in the topology defined using the SlurmTopology node extra setting. The SlurmTopology setting is described earlier on. The concatenation starts in order of parent switches first. Thus, switch switch02 that is directly connected to switch switch01 is named switch01-switch02 in topology.conf.

  Example

  ```
  [root@basecm11 ~]# cmsh
  [basecm11]% device
  [basecm11->device]% set node001 -e SlurmTopology switch01 switch03
  [basecm11->device*]% set node002 -e SlurmTopology switch01 switch03
  [basecm11->device*]% set node002 -e SlurmTopology switch01 switch03
  [basecm11->device*]% set node003 -e SlurmTopology switch01 switch03
  [basecm11->device*]% set node003 -e SlurmTopology switch01 switch03
  [basecm11->device*]% set node004 -e SlurmTopology switch02 switch03
  [basecm11->device*]% set node005 -e SlurmTopology switch02 switch03
  [basecm11->device*]% set node006 -e SlurmTopology switch02 switch03
  [basecm11->device*]% commit
  ```

  The preceding results in the following content within topology.conf:

  Example

  ```
  SwitchName=switch03 Switches=switch03-switch[01,02]
  SwitchName=switch03-switch01 Nodes=node[001-003]
  SwitchName=switch03-switch02 Nodes=node[004-006]
  ```

3. The blocksettings submode under topologysettings is where the topology/block plugin settings are located:

   Example

   ```
   [basecm11->wlm[slurm]->topologysettings]% blocksettings
   [basecm11->wlm[slurm]->topologysettings->blocksettings]% show
   Parameter                      Value
   ------------------------------ ------------------------------------------------
   Block entity                   NodeGroup
   Block sizes                    2 4
   ```

   The parameters of the topology/block plugin are:

   - Block entity: what BCM entity type defines a block. Possible entity type values are:

- nodegroup: if a node is in several nodegroups then it goes to the first block
- rack: if a pair of racks is configured as twins, then nodes from both racks go to a single block

Nodes with a slurmclient role that belong to the specified entity are grouped in a single block. For instance, if two nodegroups are defined in BCM as:

- ng1: a nodegroup that includes node001 and node002, both with the slurmclient role and
- ng2: a nodegroup that includes node003 and node004, both with the slurmclient role

then topology.conf is updated with the following lines:

Example

```
BlockName=ng1 Nodes=node[001,002]
BlockName=ng2 Nodes=node[003,004]
```

- Block sizes: corresponds to the parameter in topology.conf with the same name. It displays a list of the planning base block size, alongside any higher-level block sizes that would be enforced. More details can be found in man topology.conf.5.

4. Topograph settings can be found within the topographsettings submode, under topologysettings. The settings are described further on.

Topograph service integration: A topograph in real life can be a monument that shows important geographical properties (for example altitude, latitude, roads, and so on) in a very visible location (typically on a hilltop). In BCM, Topograph is used to mean the topological record that is visible to the cluster manager in BCM. Topograph runs as a system service, and BCM fetches topology information from the service when required, updating topology.conf as needed.

Currently, Topograph, when used with cloud services, supports only cloud GPU nodes that are supported by the topology API that the cloud providers provide. The nodes in the cloud must exist—that is, it must be possible to stop them—during topology generation. If Topograph cannot get the topology for a node, then the node in topology.conf is put under a special fake switch (if node is set to tree topology) or block (if the node is set to block topology).

Basic Topograph configuration with some default settings is carried out with cm-wlm-setup (page 415), as part of Topology configuration.

The integration between BCM and Topograph can then be further configured within the topographsettings submode:

Example

```
[basecm11->wlm[slurm]->topologysettings]% topographsettings
[basecm11->wlm[slurm]->topologysettings->topographsettings]% show
Parameter                      Value
------------------------------ ------------------------------------------------
Hostname                       localhost
Config path                    /etc/topograph/topograph-config.yaml
HTTP parameters
Pass cloud credentials         yes
Fetch timeout                  20
Fetch attempts                 12
Fetch wait time                15
```

The Topograph settings are:

- Hostname: the hostname where these settings run

- Config path: a path for the YAML configuration of the Topograph settings
- HTTP parameters: used to pass a string of HTTP parameters to the HTTP API service
- Pass cloud credentials: if set to yes, then cloud service provider credentials are passed on to the topology generator
- Fetch timeout: the Topograph API timeout value (in seconds)
- Fetch attempts: the number of attempts to carry out when trying to get a topology from Topograph
- Fetch waittime: the wait time between fetch attempts (in seconds)

and there are also extra settings, which are used with default values when configuring Topograph manually:

- HttpPort: the tcp port used by Topograph to listen on (default: 49021)
- HttpSSL: is Topograph using SSL? (default: true)
- RequestAggregationDelay: delay for request aggregation (default: 15s)
- PageSize: number of results per API call (default: 100)
- SSLCertPath: server certificate path (default: /etc/topograph/ssl/server-cert.pem)
- SSLCertKeyPath: server certificate key path (default: /etc/topograph/ssl/server-key.pem)
- SSLCACert: certificate authority certificate path (default: /etc/topograph/ssl/ca-cert.pem)
- version4: when set to true, BCM uses the Topograph 4 API. This setting requires the topograph4 package (see "Using Topograph 4" below).

The service has its own configuration file, topography-config.yaml, which is managed by BCM. The administrator can modify it manually if necessary, to put in settings that are outside of the group of standard and extra topograph settings listed in the preceding text.

If standard or extra topography settings in the navigation path:

cmsh > wlm > topologysettings > topographsettings

are set, then the topography-config.yaml file synchronizes its values from there. That is, no manual changes are typically needed in BCM-managed clusters for the BCM-managed variables.

If the node running the Topograph service is not the node running the Slurm server, then all certificate paths must be in a shared location, such as /cm/shared, and the token UPDATE_TOPOLOGY_TOKEN must be added to the profile node. Profiles are described in section 6.5.

All certificates are managed by BCM and must share the same certificate authority file path.

BCM uses the script /cm/local/apps/cmd/scripts/topograph.certs.sh to generate all certificates. An alternative certificate generator can be set by the administrator using the AdvancedConfig directive (page 925) TopographCertificatesScript to set a custom script path, to use the same environment variables and options.

If no automatic script generation is wanted then it can be disabled by setting the AdvancedConfig directive TopographSkipCertificates to yes:

```
[root@basecm11~]# cm-manipulate-advanced-config.py TopographSkipCertificates=yes
Updated: /cm/local/apps/cmd/etc/cmd.conf
```

Topology configuration with `cm-wlm-setup`: The topology source can be configured using `cm-wlm-setup` (figure 7.24), along with some settings:

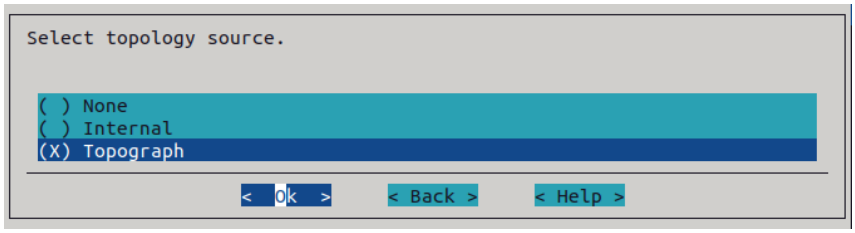


Figure 7.24: Topology source configuration in `cm-wlm-setup`

If the topology source is Topograph, then a specific node can be selected (figure 7.25):

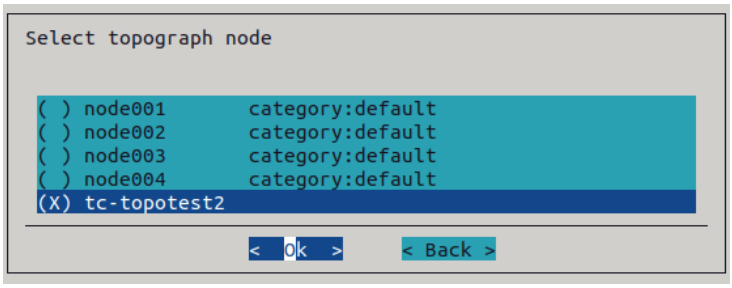


Figure 7.25: Topograph node configuration in `cm-wlm-setup`

The configuration sets all the extra values of section 7.5.1, it installs the Topograph service on the software image or in the node if necessary, and adds `UPDATE_TOPOLOGY_TOKEN` to the node profile in `cmsh`, if the Topograph node and Slurm server nodes are different.

The plugin can also be configured with `cm-wlm-setup` (figure 7.26):

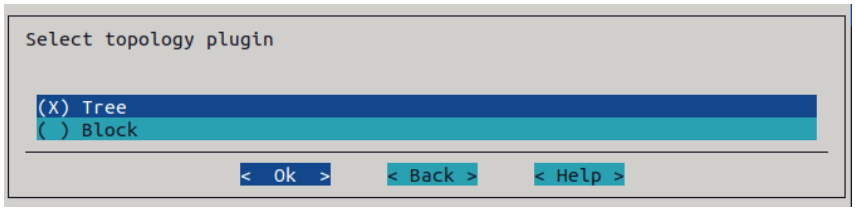


Figure 7.26: Topology plugin configuration in `cm-wlm-setup`

For more advanced and specific configurations, a further manual configuration is necessary.

Configuring topology manually: The `topograph` package is available from the BCM repositories for Ubuntu 22.04 and later, and for RHEL9-compatible distributions:

Example

```
root@basecm11:~# apt install topograph
...
Need to get 30.1 MB of archives.
After this operation, 0 B of additional disk space will be used.
...
```

After picking up the package, the service can be set up on any node. The `hostname` parameter in `topographsettings` is set to `localhost` by default, which is taken from the node that has the

slurmserver role assigned to it. The value should be the Topograph server hostname or IP address, which means that if the Topograph service is running on a node without the slurmserver role, then the value must be manually adjusted.

Typically, since slurmserver usually runs on the head node, topograph also runs on the head node, and if the cluster is an HA cluster (Chapter 15) then it should be set up on both head nodes.

The service can be set up on the head node(s) in BCM using the services submode of device mode:

Example

```
[basecm11->device[basecm11]]% services
[basecm11->device[basecm11]->services]% add topograph
[basecm11->device*[basecm11*]->services*[topograph*]]% set autostart yes
[basecm11->device*[basecm11*]->services*[topograph*]]% set monitored yes
[basecm11->device*[basecm11*]->services*[topograph*]]% set managed yes
[basecm11->device*[basecm11*]->services*[topograph*]]% commit
[basecm11->device[basecm11]->services[topograph]]% status
Service                      Status
---------------------------- -----------
topograph                    [   UP   ]
                    repeated for the other head node if using HA
```

The certificates for the topograph service are generated automatically. The certificates are used by BCM to carry out service API calls.

After restarting CMDaemon on nodes with the slurmserver role with:

```
systemctl restart cmd
```

a topology.conf file is generated in /cm/shared/apps/slurm/etc/<*slurmcluster*>/topology.conf.

A Slurm topology configuration for the Slurm topology/tree plugin (https://slurm.schedmd.com/topology.conf.html) is generated if a Slurm compute node is affected by:

- poweroff, shutdown, or power up
- creation or termination in cloud nodes
- a Slurm-related configuration change due to a BCM configuration change

Using Topograph 4: From BCM 11.34, Topograph 4 is available via the topograph4 package. The topograph4 and topograph packages conflict and cannot both be installed on the same software image or node. The topograph package (Topograph 3.x) is installed by default.

To use Topograph 4, both of the following steps are required:

1. The topograph package is removed and topograph4 is installed:

   For Ubuntu distributions:

   Example

   ```
   root@basecm11:~# apt remove topograph
   root@basecm11:~# apt install topograph4
   ```

   For RHEL-based distributions, yum can resolve the package conflict directly with the --allowerasing option, which removes topograph as part of installing topograph4:

   Example

   ```
   [root@basecm11 ~]# yum install --allowerasing topograph4
   ```

2. The version4 extra value is set in topographsettings:

Example

```
[basecm11->wlm[slurm]->topologysettings->topographsettings]% set -e version4 true
[basecm11->wlm*[slurm*]->topologysettings*->topographsettings*]% commit
```

Both steps are required. Installing topograph4 without setting version4 to true, or setting version4 to true while the topograph package is still installed, causes the Topograph service to fail.

The version4 extra value is a setting on the Slurm WLM instance, not a per-node setting, so it only needs to be set once, regardless of how many nodes run Topograph.

topograph4 must be installed on every node on which topograph is currently running. On an HA cluster (Chapter 15), that means both head nodes, matching the placement described in the preceding text. If Topograph runs from a software image rather than only on head nodes, topograph4 must be installed into the image (section 9.4.1) and the standard updateprovisioners (section 5.2.4) and imageupdate (section 5.6.2) commands must be used to propagate the change to the running nodes, rather than it being installed directly on a running node.

Installing topograph4 replaces the systemd unit that was installed by the topograph package. As with other BCM-managed services whose systemd unit changes (for example, the slurmd service after a Slurm software-image upgrade, section 7.5.1), the systemd unit files must be reloaded and the service restarted on every node where topograph4 was just installed, following the placement described in the preceding paragraph:

Example

```
[root@basecm11 ~]# systemctl daemon-reload
cmsh -c "device services basecm11; use topograph; restart"
```

The FakeNodes and FakeNodesPool extra settings, which were special values used in earlier versions of some Slurm clusters, are no longer needed in recent versions of Slurm, and are not supported in Topograph 4. If these settings are configured when version4 is active, BCM logs a warning and ignores them.

To revert to Topograph 3, the version4 extra value must be cleared, and the topograph package reinstalled on every node currently running topograph4 – directly on the node if it was installed there (for example, a head node), or in the software image, followed by updateprovisioners and imageupdate, if it was installed via a software image.

For cloud service providers managed by BCM there are some extra considerations for the topograph service:

- For AWS clouds the compute nodes must be running (exist), and must have a flavor that supports the AWS topology API (https://docs.aws.amazon.com/AWSEC2/latest/UserGuide/ec2-instance-topology.html). One of the supported flavors is the p5.48xlarge instance currently used by DGX Cloud.

  Requirements:

  - To access the AWS instance topology API using the topograph service, a prerequisite for an IAM identity is to have permissions for ec2:DescribeInstanceTopology. This permission can be enabled through an AWS-managed policy called AmazonEC2ReadOnlyAccess.
  - Topograph needs credentials to perform the AWS API call.
    - If the administrator sets the value of Pass cloud credentials to yes, then BCM sends the credentials to the topograph service on each topology fetch.

    * If the administrator sets the value of `Pass cloud credentials` to `no` in BCM, then the profile of the EC2 instance where `topograph` runs has an IAM role with the `AmazonEC2ReadOnlyAccess` policy enabled. AWS EC2 instances support an IMDS API that provides metadata related to the instance itself. One of the features of this API is the support of a profile role for an instance.

- For OCI clouds the nodes must exist in OCI, and
  - the administrator can set the value of `Pass cloud credentials` to `yes`

  or
  - the administrator can set the value of `Pass cloud credentials` to `no` and specify some named dynamic group (`https://docs.oracle.com/en-us/iaas/Content/Identity/Tasks/managingdynamicgroups.htm`) policies in the tenancy.

    Each named dynamic group must have permissions to inspect certain resources. Additionally, the special BCM `thebcmnonprod-topo-dg` dynamic group must have various permissions set to use a variety of resources. The permissions can be set according to the OCI policy syntax (`https://docs.oracle.com/en-us/iaas/Content/Identity/Tasks/callingservicesfrominstances.htm#Writing`) as follows:

    ```
    Allow dynamic-group <dynamic_group_name> to inspect compute-capacity-topologies in tenancy
    Allow dynamic-group <dynamic_group_name> to inspect compute-bare-metal-hosts in tenancy
    Allow dynamic-group <dynamic_group_name> to inspect compute-hpc-islands in tenancy
    Allow dynamic-group <dynamic_group_name> to inspect compute-network-blocks in tenancy
    Allow dynamic-group <dynamic_group_name> to inspect compute-local-blocks in tenancy
    ```

- For GCP clouds the requirement is that the placement policy for VMs should be `compact`.

After the Topograph package is installed, the service started, and the administrator has met the requirements for the cloud topology API, then the `topologyplugin` and `topologysource` values must be set:

Example

```
[basecm11]% wlm use slurm
[basecm11->wlm[slurm]]% topologysettings
[basecm11->wlm[slurm]->topologysettings]% set topologyplugin tree
[basecm11->wlm[slurm]->topologysettings*]% set topologysource topograph
[basecm11->wlm[slurm]->topologysettings*]% commit
```

If `topologysource` is set to a value of `none`, then `topology.conf` content is not updated by BCM. Setting a value of `topograph` causes the Slurm topology to be fetched from the `topograph` service. The `topograph` service currently supports only the tree and block plugins.

Two additional `cmsh` commands allow testing:

- The `topology update` command starts a topology update in the background.
- The `topology print` command prints the generated content for Slurm `topology.conf` to STDOUT. This is a synchronous operation, which means that it can freeze for 10-15 seconds when Topograph is in use.

Example

```
[root@basecm11 ~]# cmsh
[basecm11]% wlm use slurm
[basecm11->wlm[slurm]]% topology update
```

```
Topology update initiated
[basecm11->wlm[slurm]]% topology print
SwitchName=nn-468b60a0020d2b170 Switches=nn-7a2f7858c1f7d5cd1
SwitchName=nn-7a2f7858c1f7d5cd1 Switches=nn-73d157eb3d683970e,nn-400810bf9f5e465d6
SwitchName=nn-400810bf9f5e465d6 Nodes=cnode001
SwitchName=nn-73d157eb3d683970e Nodes=cnode002
```

The topograph service logs can be seen with the systemd command: `journalctl -u topograph -f`

Prometheus exporter sampling of the topograph service can be configured in CMDaemon. The topograph service sends out Prometheus metrics, which can be added to BCM as follows:

Example

```
[root@basecm11 ~]# cmsh
[basecm11]% monitoring setup
[basecm11->monitoring->setup]% add prometheus topograph
[basecm11->monitoring->setup*[topograph*]]% set urls https://localhost:49021/metrics
[basecm11->monitoring->setup*[topograph*]]% set -e NoPostAllowed yes
[basecm11->monitoring->setup*[topograph*]]% set https yes
[basecm11->monitoring->setup*[topograph*]]% set cacertificatepath /etc/topograph/ssl/ca-cert.pem
[basecm11->monitoring->setup*[topograph*]]% set privatekeypath /etc/topograph/ssl/server-key.pem
[basecm11->monitoring->setup*[topograph*]]% set certificatepath /etc/topograph/ssl/server-cert.pem
[basecm11->monitoring->setup*[topograph*]]% nodeexecutionfilters
[basecm11->monitoring->setup*[topograph*]->nodeexecutionfilters]% active
Added active resource filter
[basecm11->monitoring->setup*[topograph*]->nodeexecutionfilters]% commit
```

If, in the following session, the number of measurables is not zero, then it is a confirmation that the metrics are consumed by BCM:

Example

```
[root@basecm11 ~]# cmsh
[basecm11]% monitoring setup
[basecm11->monitoring->setup]% use topograph
[basecm11->monitoring->setup[topograph]]% show | grep measurables -i
Measurables                    26: comma-separated list of measurables
```

Slurm drain reason policy: CMDaemon allows nodes to be drained:

- either by user request, using the `drain` command (section 7.7.4) or
- automatically, when a monitoring trigger (section 10.4.5) is configured. For example, if some healthcheck fails, then the appropriate node can be drained automatically.

The administrator can also manually drain a node outside of `cmsh`, using the Slurm `scontrol` command from the terminal, as described in `man scontrol.1`.

In all these cases the drain reason can be specified. By default, if a node is already drained, then a second drain command with a new drain reason replaces the old drain reason.

The behavior can be configured with the parameter `Drain reason policy`, within the Slurm cluster settings. The `Drain reason policy` can take one of the following (case-insensitive) values:

- `REPLACE`: the old drain reason(s) is replaced by a new one
- `APPEND`: the new drain reason is separated by a comma, and appended to the existing one(s).

- SKIP:
  - If a drain reason already exists, then setting the new drain reason is skipped.
  - If no drain reason already exists, then the new drain reason is applied.

In the following example, the drain reason policy is set to APPEND. This means that CMDaemon always appends a new drain reason to any existing ones.

Example

```
[root@basecm11 ~]# cmsh
[basecm11]% wlm use slurm
[basecm11->wlm[slurm]]% set drainreasonpolicy APPEND
[basecm11->wlm[slurm*]]% commit
```

It is also possible to temporarily set an append policy in cmsh when draining a node, by using the + operator as a prefix to the drain reason:

Example

```
[root@basecm11 ~]# cmsh
[basecm11]% device use node001
[basecm11->device[node001]]% drain --reason="+this will be appended regardless of the current policy"
```

Slurm services restart or reconfiguration conditions:

- When parameters in the Slurm configuration files are changed by BCM using CMDaemon, typically by actions of cmsh or Base View, then Slurm services automatically restart, and scontrol reconfigure is automatically run, as required.
- When parameters in the Slurm configuration files are changed directly, then BCM does not detect a change automatically, and therefore does not automatically restart or reconfigure Slurm.

Slurm subsystems are restarted and reconfigured based on the following list of conditions:

- slurmctld and slurmd are restarted if:
  1. any of the following global slurm.conf parameters managed by CMDaemon are changed:
     - SelectType
     - SelectTypeParameters
     - GresTypes
     - StateSaveLocation
     - SchedulerType
     - SchedulerParameters
     - SlurmctldParameters
  2. gres.conf is changed (by CMDaemon)
  3. oci.conf (https://slurm.schedmd.com/oci.conf.html) is changed (by CMDaemon). In cmsh this typically happens using the ocisettings submode under the wlm mode.
  4. nodes are changed (by CMDaemon) in slurm.conf (nodes are renamed, added, or removed).
- An scontrol reconfigure is carried out if:
  1. any Slurm parameters managed by BCM, that are not mentioned in the preceding list of conditions, are changed. This is true also for the Partition and Node parameters within the slurm.conf file, except for the NodeName and PartitionName parameters specifically.
  2. cgroups.conf is changed (by CMDaemon)
  3. topology.conf is changed (by CMDaemon)

A restart or a reconfigure is not done by CMDaemon under other conditions.

Slurm resource consumption monitoring configuration: Slurm allows the configuration of how consumable resources (core, CPU, memory, etc) are tracked and shared among jobs on a node. At the time of writing of this section (November 2022), `cmsh` or Base View can be used to configure the tracking and sharing of resources as in the following text.

- `SelectType`: this plugin configures the algorithm used to select resources for jobs. By default, NVIDIA Base Command Manager configures the `select/cons_tres` plugin. This is an advanced version of `select/cons_res` plugin, and it also allows GPUs to be tracked separately from other consumable resources.

  Further details on `cons_res` can be found in the Slurm documentation at `https://slurm.schedmd.com/cons_res.html` and `https://slurm.schedmd.com/cons_res_share.html`.

  The `SelectType` parameter is configured in `cmsh` or Base View, from within the Slurm cluster configuration settings:

  Example

  ```
  [root@basecm11 ~]# cmsh
  [basecm11]% wlm use slurm
  [basecm11->wlm[slurm]]% set selecttype select<TAB><TAB>
  select/cons_res   select/cons_tres  select/linear
  [basecm11->wlm[slurm]]%
  ```

- `SelectTypeParameters`: allows the configuration of the resource selection plugin that is specified for the `SelectType` parameter. The possible values depend on the plugin type. For `select/cons_tres`, the values follow the pattern:

  - `CR_Memory` or
  - `CR_`<*COMPUTE_UNIT*>`[_Memory]` where <*COMPUTE_UNIT*> takes one of the following values:
    - `Core`
    - `CPU`
    - `Socket`
    - `Board`

  The value of `SelectTypeParameters` specifies if the <*COMPUTE_UNIT*> resource is tracked, or if the <*COMPUTE_UNIT*> resource is tracked together with memory, or if only memory is tracked.

  There are also a few special values that can be applied to `SelectTypeParameters`:

  - `CR_ONE_TASK_PER_CORE`: allocate one task per core (used only with the `select/linear` plugin)
  - `CR_CORE_DEFAULT_DIST_BLOCK`: allocate cores within a node using a block distribution
  - `CR_LLN`: schedule resources to jobs on the least loaded nodes
  - `CR_Pack_Nodes`: if a job allocation contains more resources than will be used for launching tasks, then rather than distributing a job's tasks evenly across its allocated nodes, pack them as tightly as possible on these nodes.

- `OverSubscribe` (page 203):

  - if the `select/linear` plugin is used, then the `OverSubscribe` parameter controls whether or not the nodes are shared among jobs

  - if the `select/cons_res` or `select/cons_tres` plugins are used, then the `OverSubscribe` parameter controls whether or not the configured consumable resources are shared among jobs. For these plugins, when a consumable resource such as a core, socket, CPU, or other, is shared, it means that more than one job can be assigned to it. The parameter is set per Slurm partition (queue) and accepts the following values:
    * `EXCLUSIVE`: allocate entire node for a job;
    * `FORCE`: makes all resources (except for GRES) in the Slurm partition available for oversubscription without any means for users to disable it (may be followed with a colon and maximum number of jobs in running or suspended state);
    * `YES`: makes all resources (except for GRES) in the Slurm partition available for sharing upon request by the job (may be followed with a colon and maximum number of jobs in running or suspended state);
    * `NO`: no resource is allocated to more than one job.

- `AccountingStorageTRES`: A comma-separated list of resources that the administrator wants to track on the cluster. By default the following resources (TRES) are tracked:
  - billing
  - CPU
  - energy
  - memory
  - node
  - fs/disk
  - pages
  - vmem

  These default TRES cannot be disabled, but only appended to. Resources use is recorded when used on the cluster. If GPUs of different types are tracked, then job requests with matching type specifications are recorded.

  Example

  If:
  `AccountingStorageTRES=gres/gpu:tesla,gres/gpu:volta` is set
  then:
  `gres/gpu:tesla` and `gres/gpu:volta` track only jobs that explicitly request those two GPU types.

Slurm REST API: `slurmrestd` is Slurm's REST API daemon. It is installed with the BCM package `slurm25.05-slurmrestd`, or `slurm25.11-slurmrestd`. The distribution package, `slurm-slurmrestd`, should not be installed, and indeed its installation is not suggested by the package manager in a cluster that is correctly configured.

Slurm REST API package installation and version matching: If installing the REST API package, then care must be taken to match the versions exactly to its sibling Slurm packages (page 387). For example, if the already-installed packages have version 25.05:

Example

```
[root@basecm11 ~]# rpm -qa slurm*      #what are the already installed packages?
slurm25.05-25.05.8-101124_cm11.0_0639470153.x86_64
slurm25.05-perlapi-25.05.8-101124_cm11.0_0639470153.x86_64
slurm25.05-slurmdbd-25.05.8-101124_cm11.0_0639470153.x86_64
slurm25.05-slurmd-25.05.8-101124_cm11.0_0639470153.x86_64
slurm25.05-slurmctld-25.05.8-101124_cm11.0_0639470153.x86_64
slurm25.05-contribs-25.05.8-101124_cm11.0_0639470153.x86_64
slurm25.05-devel-25.05.8-101124_cm11.0_0639470153.x86_64
slurm25.05-prs-25.05.8-101124_cm11.0_0639470153.x86_64
```

and if the Slurm REST API package version to be installed is as indicated by the following:

Example

```
root@basecm11 ~]# yum info slurm25.05-slurmrestd | egrep '(Name|Release|Source)'  #what will install?
Name         : slurm25.05-slurmrestd
Release      : 102530_cm11.0_f08198ecda
Source       : slurm25.05-25.05.9-102530_cm11.0_f08198ecda.src.rpm
```

then it means that all already-installed Slurm packages must be upgraded to release `102530_cm11.0_f08198ecda` too.

The reason behind matching versions exactly is that the upstream versions can have significant configuration changes between patch versions (here `25.05.8` changing to `25.05.9`). These changes may result in a non-functioning workload manager. This is true, not only when considering the Slurm REST API package, but also true when considering the other Slurm packages.

The user `daemon` cannot be in the `bin` group for `slurm25.05-slurmrestd` on SLES15: Slurm 25.05 expects that the user `daemon` should be a member of only the group `daemon`. The `groups` command lists the groups that a user is in, so it can be used to check to see if the user `daemon` is in the group `bin` as well in the group `daemon` with:

Example

```
[root@basecm11 ~]# groups daemon
daemon : daemon bin
```

Here the user `daemon` is in the group `bin` and as well as in the group `daemon`. For `slurmrestd` to work properly, the group `bin` must have user `daemon` deleted from it. The `gpasswd` command can do that with:

Example

```
[root@basecm11 ~]# gpasswd -d daemon bin
```

If `slurmrestd` is running on a compute node, then the `gpasswd` command must be run in the image of the compute node instead. For example, if the image is `default-image`:

Example

```
[root@basecm11 ~]# cm-chroot-sw-img /cm/images/default-image gpasswd -d daemon bin
```

The `imageupdate` command can then be run so that the change is picked up by the node.

The `gpasswd` command should be run for any group, other than group `daemon`, that user `daemon` is a member of.

Slurm REST API and authentication: The slurmrestd service can be configured with JWT (JSON Web Token) authentication as follows:

1. The JWT key is generated in the Slurm cluster configuration directory, and its ownership and permissions are then changed:

   Example

   ```
   [root@basecm11 ~]# module load slurm  # Loads SLURM_CONF environment variable from modulefile
   [root@basecm11 ~]# JWT_KEY=`dirname $SLURM_CONF`/jwt.key
   [root@basecm11 ~]# install -m 0600 -o slurm -g slurm <(dd if=/dev/random bs=32 count=1) $JWT_KEY
   ```

   The location of the key can differ, but user `slurm` must have read access to the file.

2. The JWT plugin is configured in `slurm.conf` for the Slurm cluster instance. For example, for a Slurm cluster instance <*cluster name*>:

   Example

   ```
   AuthAltTypes=auth/jwt
   AuthAltParameters=jwt_key=/cm/shared/apps/slurm/etc/<cluster name>/jwt.key
   ```

   The `slurmctld` service is restarted to apply the plugin settings. For example, if it is running on the head node:

   ```
   [root@basecm11 ~]# systemctl restart slurmctld.service
   ```

3. A `systemd` drop-in file is created for the new `slurmrestd` service:

   Example

   ```
   [root@basecm11 ~]# slurmrestapidir="/etc/systemd/system/slurmrestd.service.d"
   [root@basecm11 ~]# mkdir $slurmrestapidir
   [root@basecm11 ~]# echo "[Service]" > $slurmrestapidir/99-cmd.conf
   [root@basecm11 ~]# echo "Environment=SLURM_CONF=$SLURM_CONF" >> $slurmrestapidir/99-cmd.conf
   ```

   Future versions of BCM may automatically create or update the file `99-cmd.conf`. It is therefore recommended to use that file name as a best practice for cluster administration.

4. The `slurmrestd` service is configured within `cmsh` so that it can be monitored by BCM:

   Example

   ```
   [root@basecm11 ~]# cmsh
   [basecm11]% device services master
   [basecm11->device[basecm11]->services]% add slurmrestd
   [basecm11->device*[basecm11*]->services*[slurmrestd*]]% set monitored yes
   [basecm11->device*[basecm11*]->services*[slurmrestd*]]% set autostart yes
   [basecm11->device*[basecm11*]->services*[slurmrestd*]]% commit
   [basecm11->device[basecm11]->services[slurmrestd]]%.
   ```

5. A ping-like `curl` check can be run with custom extra headers to ensure that the Slurm REST API service is running properly, as follows:

Example

```
[root@basecm11 ~]# export $(scontrol token username=cmsupport)
[root@basecm11 ~]# curl 0.0.0.0:6820/openapi \
-H "X-SLURM-USER-TOKEN: $SLURM_JWT" \
-H "X-SLURM-USER-NAME: cmsupport"
  % Total    % Received % Xferd  Average Speed     Time     Time     Time  Current
                                 Dload  Upload    Total    Spent     Left  Speed
  0     0    0        0 0     0        0     0 --:--:-- --:--:-- --:--:--        0
{
  "tags": [
    {
      "name": "slurm",
      "description": "methods that query slurmdbd"
    },
    {
      "name": "openapi",
      "description": "methods that query for OpenAPI specifications"
    }

[...]
```

The `slurmrestd` daemon configuration is specified by its command line arguments. These can be made permanent in `/etc/sysconfig/slurmrestd`.

GPU autodetection: Slurm supports GPU autodetection from version 20 onward, using the `AutoDetect` flag in `gres.conf`.

Slurm GPU autodetection allows AMD, Intel, and NVIDIA GPUs to have their GPU parameters be set automatically, if autodetection is also enabled in `gres.conf`. Slurm GPU autodetection removes the need for explicit GPU configuration in `gres.conf`.

The number of GPUs must however still be specified explicitly by the administrator. This is because at the time of writing (October 2024) Slurm has no GPU autodetection by default.

In BCM, the Slurm GPU hardware detection setting has 7 options, as covered on page 405.

- Setting the `bcm` option for Slurm GPU autodetection removes the need to specify any further GPU autodetect option.
- Setting a GPU vendor (NVIDIA, Intel, AMD) option for Slurm GPU autodetection means that the `Gres=...` parameter in `NodeName` lines in `slurm.conf` is still required in order to tell `slurmctld` how many GRES to expect. This means that the administrator must define GPUs in BCM, in the `genericresources` mode of the `slurmclient` role, if that has not already been specified during `cm-wlm-setup`.

  However there is no need in this case to specify all the details. It is enough to add a single generic resource named `gpu`, and to specify the number of such GPUs. In order to skip adding the generic resources to `gres.conf`, while still allowing GRES information to be added the `slurm.conf` file, the flag `AddToGresConfig` should be set to `no` in the `genericresource` entity. For example:

Example

```
[basecm11]% configurationoverlay use slurm-client
[basecm11->configurationoverlay[slurm-client]]% roles
[basecm11->configurationoverlay[slurm-client]->roles]% use slurmclient
[...[slurmclient]]% genericresources
[...[slurmclient]->genericresources]% add autodetected-gpus
[...[slurmclient*]->genericresources*[autodetected-gpus*]]% set name gpu
[...[slurmclient*]->genericresources*[autodetected-gpus*]]% set count 8
[...[slurmclient*]->genericresources*[autodetected-gpus*]]% set addtogresconfig no
[...[slurmclient*]->genericresources*[autodetected-gpus*]]% commit
[...[slurmclient]->genericresources[autodetected-gpus]]% wlm use slurm
[basecm11->wlm[slurm]]% set gpuautodetect nvml
[basecm11->wlm*[slurm*]]% commit
[basecm11->wlm[slurm]]%
```

Initial generic resources configuration for Slurm GPU autodetection can also be done during Slurm setup by `cm-wlm-setup`. If GPU configuration is chosen in the TUI screen of `cm-wlm-setup` (figure 7.27):

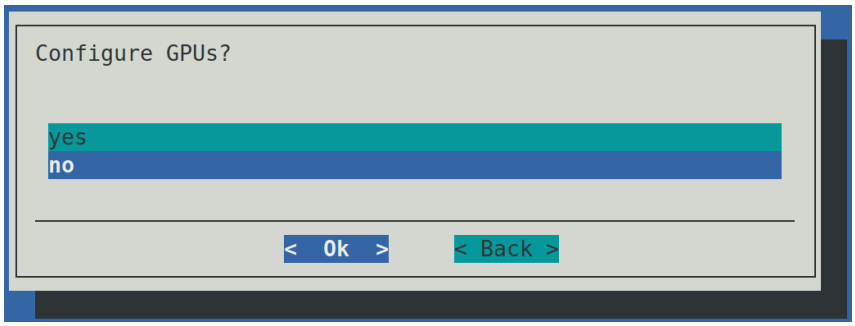


Figure 7.27: The screen prompting for GPU configuration in `cm-wlm-setup`

then the TUI allows Slurm GPU autodetection options to be selected (figure 7.28):

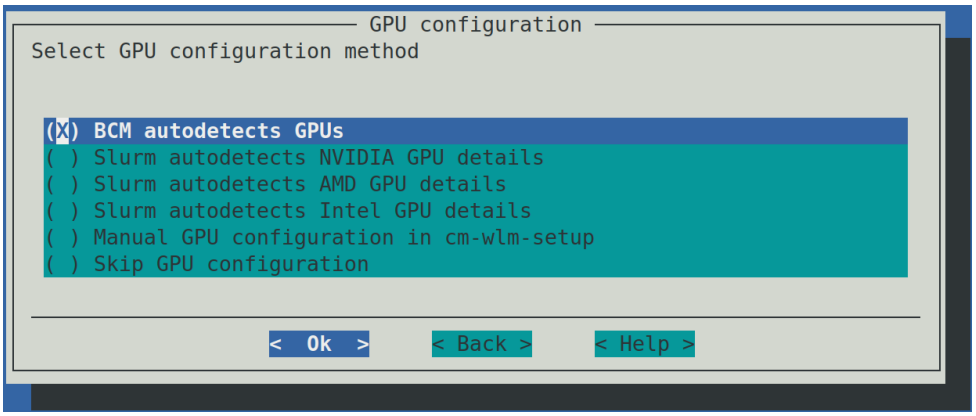


Figure 7.28: The `GPU configuration` screen of `cm-wlm-setup`

The options are the same as those described in the list of 7 options for `GPU auto detect` (page 406). The options result in the following actions:

- `BCM autodetects GPUs`: BCM automatically detects numbers and types of GPUs on the selected nodes and configures `slurm.conf` appropriately. The administrator is not asked to configure anything. This option does not work if the nodes have GPUs from a mix of vendors.
- `Slurm autodetects` *<vendor>* `GPU details`: the administrator is prompted to specify numbers and types of GPUs later on. These values are then added to the `Gres` parameter of `NodeName` lines

in slurm.conf, while Slurm autodetects the rest of the GPU details.

- Manual GPU configuration in cm-wlm-setup: the administrator is prompted to enter the GPU parameters manually. The manual entries are then converted to appropriate lines in gres.conf.

Slurm GPU cluster settings: When a GPU is configured the following parameters can be configured with BCM:

Base View Option	Slurm Object Value Within WLM Mode Of cmsh	Description
Select Type	SelectType	Identifies the type of resource selection algorithm to be used. Changing this value causes the slurmctld daemon to be restarted. Default: Consumable trackable resource:select/cons_tres
Select Type Parameters	SelectTypeParameters	Parameters for the resource selection algorithm. Acceptable values for these parameters depend upon the algorithm selected. Default: Core as a consumable resource: CR_Core
Accounting Storage TRES	AccountingStorageTRES	Comma-separated list of resources to be tracked on the cluster, requested by an sbatch/srun job on submission. Default: GPU as a generic resource: gres/gpu

Example

```
[basecm11->wlm[slurm]]% get selecttype
select/cons_tres
```

MIG and Slurm: NVIDIA's Multi-Instance GPU (MIG) technology is a way to optimize GPU use. MIG allows a single GPU to be partitioned into up to 7 separate logical GPU instances, each providing separate GPU resources to user jobs. Slurm is MIG-capable since version 21.08.

BCM automates the configuration of MIG profiles as GPU generic resource types. This means that a user can request GPUs by type, and Slurm then tracks the MIG profiles usage.

MIG detection on Slurm compute nodes (nodes with the slurmclient role assigned) means that the slurm.conf and gres.conf files are automatically configured by CMDaemon as follows:

- slurm.conf:

  1. For each node, NodeName parameter lines have a Gres specification (generic resource type specification) with the MIG profile names appended as strings. The strings take the format:

     gpu:<*profile*>:<*number*>

     where <*number*> is the number of instances with the profile name of gpu:<*profile*>. For example:

     (a) A GPU on node001 is partitioned into two MIG instances, one with a profile name of 3g.40gb, and the other with a profile name of 4g.40gb:

     Example

```
NodeName=node001 ... Gres=gpu:3g.40gb:1,gpu:4g.40gb:1 ...
```

(b) A GPU on node001 is partitioned into two MIG instances, both with same profile name 3g.20gb. Also present is a regular GPU, a40, that is configured manually as a generic resource:

Example

```
NodeName=node001 ... Gres=gpu:a40:1,gpu:3g.20gb:2 ...
```

2. The GresTypes parameter line has the MIG profiles from all the compute nodes.

   Example

   ```
   GresTypes=3g.40gb,4g.40gb,gpu
   ```

3. The AccountingStorageTRES parameter line includes the new generic resource per MIG profile.

   Example

   ```
   AccountingStorageTRES=gres/gpu,gres/gpu:3g.40gb,gres/gpu:4g.40gb
   ```

- gres.conf: the AutoDetect parameter is set to NVML for the nodes where MIGs are detected.

  Example

  ```
  NodeName=node[001-016] AutoDetect=NVML
  ```

  In this case Slurm detects the MIG details (links, file, cores, and so on) automatically.

If the GPUs are repartitioned by BCM, then the Slurm configuration is by default automatically updated with the new GPU types according to the new MIG profiles. The default value of yes can be modified with the configuremigs parameter in Slurm cluster settings:

Example

```
[basecm11->wlm[slurm]]% set configuremigs no; commit
```

The administrator can validate that the MIG devices are correctly recognized by Slurm, by checking the Gres line output of the scontrol show node command:

Example

```
[root@basecm11 ~]# scontrol show node node001 | grep "Gres="
   Gres=gpu:4g.40gb:1(S:0),gpu:3g.40gb:1(S:0)
```

The presence of a socket value such as S:0 indicates that Slurm detected the MIG details on the node as expected.

Slurm node settings: The parameter value for a Slurm option in slurm.conf is set by CMDaemon, if its value is not: 0.

A parameter value of 0 means that the default values of Slurm are used. These usually have the value: 0.

The advanced options that CMDaemon manages for Slurm are:

Base View Option	Slurm Option	Description
Features	Feature=<*string*> entry in the file slurm.conf	Arbitrary strings can be entered to indicate some characteristics of a node, one string per entry. For example: text1 text2 and so on. These become part of the: Feature=text1,text2... attribute to the NodeName=<*node name*> entry line in slurm.conf, as indicated in man slurm.conf.5. The strings also become added attributes to the GresTypes entry of that file. Default: blank.
Slots	CPU	Number of logical processors on the node. For Slurm 20 and beyond, CMDaemon detects the number of CPU cores, and sets procs=*number of cores* using slots autodetection. Default for Slurm prior to version 20: 0
Sockets	Sockets	Processor chips on node. If this is defined, then SocketsPerBoard must not be defined. Default: 0
Cores per socket	CoresPerSocket	Number of cores per socket. Default: 0
ThreadsPerCore	ThreadsPerCore	Number of logical threads for a single core. Default: 0
Boards	Boards	Number of baseboards in a node. Default: 0
SocketsPerBoard	SocketsPerBoard	Number of processor chips on baseboard. If this is defined, then Sockets must not be defined. Default: 0
RealMemory	RealMemory	Size of real memory on the node, MB. Default: 0
NodeHostname	NodeHostname	Default: as defined by Slurm's NodeName parameter.

...continues

...continued

Base View Option	Slurm Option	Description
NodeAddr	NodeAddr	Default: as set by Slurm's NodeHostname parameter.
State	State	State of the node with user jobs. Possible Slurm values are: DOWN, DRAIN, FAIL, FAILING, and UNKNOWN. Default: UNKNOWN
Weight	Weight	The priority of the node for scheduling. Default: 0
Port	Port	Port that slurmd listens to on the compute node. Default: as defined by SlurmdPort parameter. If SlurmdPort is not specified during build: Default: 6818.
TmpDisk	TmpDisk	Total size of Slurm's temporary filesystem, TmpFS, typically /tmp, in MB. TmpFS is the storage location available to user jobs for temporary storage. Default: 0
Options	extra options	Extra options that are added to slurm.conf

Further Slurm documentation is available:

- in man pages under /cm/local/apps/slurm/current/man/
- as HTML documentation in the version dependent directory of the form:
  /cm/local/apps/slurm/current/share/doc/slurm-<*version*>/html, or
- at the Slurm website at http://slurm.schedmd.com/documentation.html

Slurm is set up with reasonable defaults, but administrators familiar with Slurm can reconfigure the configuration file using a web browser. The web browser can be used with the HTML documentation of the installed Slurm version, in the version dependent directory of the form:
/cm/local/apps/slurm/current/share/doc/slurm-<*version*>/html/
The choice of JavaScript-based configuration generators is then:

- configurator.easy.html: for a simplified configurator
- configurator.html: for a full version of the configurator.

If the configuration file becomes mangled beyond repair, the original default can be regenerated once again by re-installing the Slurm package, then running the script /cm/local/apps/slurm/current/scripts/cm-restore-db-password, and then running cm-wlm-setup. Care must be taken to avoid duplicate parameters being set in the configuration file—slurmd may not function correctly in such a configuration.

Running Slurm

Slurm can be disabled and re-initialized with the cm-wlm-setup tool (section 7.3) during package installation itself.

Alternatively, role assignment and role removal can be used to adjust what nodes, if any, run Slurm. The assignment and removal of roles can be carried out from Base View (section 7.4.1) or `cmsh` (section 7.4.2).

The Slurm workload manager runs these daemons:

1. as servers:
   (a) `slurmdbd`: The database that tracks job accounting. It is part of the `slurmdbd` service.
   (b) `slurmctld`: The controller daemon. Monitors Slurm processes, accepts jobs, and assigns resources. It is part of the `slurm` service.
   (c) `munged`: The authentication (client-and-server) daemon. It is part of the `munge` service.
2. as clients:
   (a) `slurmd`: The compute node daemon that monitors and handles tasks allocated by `slurmctld` to the node. It is part of the `slurm` service.
   (b) `slurmstepd`: A temporary process spawned by the `slurmd` compute node daemon to handle Slurm job steps. It is not initiated directly by users or administrators.
   (c) `munged`: The authentication (client-and-server) daemon. It is part of the `munge` service.

Logs for the daemons are saved on the node that they run on. Accordingly, the locations are:

- `/var/log/slurmdbd`
- `/var/log/slurmd`
- `/var/log/slurmctld`
- `/var/log/munge/munged.log`

7.5.2 Installing, Configuring, And Running LSF

IBM prefers to make LSF available directly from their Passport Advantage Online website, which is why it is not available by direct selection in figure 3.9 of the *Installation Manual*.

IBM provides LSF in 2 ways.

1. IBM Spectrum LSF Suites (`https://www.ibm.com/products/hpc-workload-management?mhsrc=ibmsearch_a&mhq=lsf` (April 2024)). These are not supported by BCM.
2. IBM LSF Standard Edition v10.1 (`https://www.ibm.com/support/pages/node/631591` (April 2024)). This is supported by BCM.

   Installation is carried out with `cm-wlm-setup` as explained later on in this section 7.5.2. LSF Standard Edition 10.1 requires a licence ("entitlement") from IBM. The product is updated with patch releases. Patch releases can be interim fixes, or they can be a bundle of interim fixes, called a fix pack. The fixes for 64-bit Linux can be viewed at:

   `https://www.ibm.com/support/fixcentral/swg/selectFixes?parent=IBM%20Spectrum%20Computing&product=ibm/Other+software/IBM+Spectrum+LSF&release=10.1&platform=Linux+64-bit,x86_64&function=all`

   (April 2024), and filters can be selected to fine tune the fixes that are displayed.

   Fix packs use a versioning that follows the format:

   `10.1.0.<`*version*`>`

   where <*version*> can be a string such as:

   `14-spk-2023-Apr-build601547`

   Patches are installed by downloading the patches and installing them manually as new patches are released.

Somewhat confusingly, the archived IBM Spectrum LSF Suite version 10.2.0.9 has LSF Standard edition 10.1 as a component. The component is supported, but installation from the suite is not supported by BCM. That is, installation using the official IBM method or by carrying out a manual installation are both not supported by BCM. A cluster administrator should therefore not try to install LSF Standard Edition 10.1 from a IBM Spectrum LSF Suite release.

Installing LSF

The workload manager LSF version 10.1 is installed and integrated into NVIDIA Base Command Manager 11 with the following steps:

1. The following LSF files should be downloaded from the IBM web site into a directory on the head node:

   - Installation package: `lsf<`*lsf_ver*`>_lsfinstall_linux_<`*cpu_arch*`>.tar.Z`
   - Distribution package: `lsf<`*lsf_ver*`>_linux<`*kern_ver*`>-glibc<`*glibc_ver*`>-<`*cpu_arch*`>.tar.Z`
   - Documentation package (optional): `lsf<`*lsf_ver*`>_documentation.tar.Z`

   Here:

   - <*lsf_ver*> is the LSF version, for example: `10.1`
   - <*kern_ver*> is the Linux kernel version, for example: `2.6`
   - <*glibc_ver*> is the glibc library version, for example: `2.3`
   - <*cpu_arch*> is the CPU architecture, for example: `x86_64`

   A check should be done to ensure that the `tar.Z` files have not been renamed or had their names changed to lower case during the download, in order to avoid installation issues. All the files must be in the same directory before installation.

   In case of an existing failover setup, the installation is done on the active head node.

   A license file for LSF may also be needed for the installation, but it does not have to be in the same directory as the other `tar.Z` files.

2. The `cm-lsf` package must be installed on the head node. The `cm-lsf-client` package must be installed on both the head node and within the software images. By default they should have been already installed. If not then they can be installed manually from the BCM repository. For RHEL-based distributions the procedure looks like:

   ```
   [root@basecm11 ~]# yum install cm-lsf cm-lsf-client
   [root@basecm11 ~]# chroot <IMAGE> yum install cm-lsf-client
   ```

   For SLES distributions the procedure looks like:

   ```
   [root@basecm11 ~]# zypper install cm-lsf cm-lsf-client
   [root@basecm11 ~]# chroot <IMAGE> zypper install cm-lsf-client
   ```

   For Ubuntu distributions the procedure looks like:

   ```
   [root@basecm11 ~]# apt-get install cm-lsf cm-lsf-client
   [root@basecm11 ~]# cm-chroot-sw-img <IMAGE>
   [root@basecm11 ~]# apt-get install cm-lsf-client
   ...
   [root@basecm11 ~]# exit
   ```

The cm-lsf and cm-lsf-client packages contain a template for an environment module file, an installation configuration file, and systemd unit files. The installation configuration file may be tuned by the administrator if required. It is passed to the lsfinstall script distributed with LSF, which is executed by cm-wlm-setup during setup. To change the default values in the installation configuration file, the administrator should change the template file:

```
[root@basecm11 ~]# cd /cm/shared/apps/lsf/var/cm/
[root@basecm11 cm]# vi install.config.template
...
```

Values enclosed by a percentage sign, '%', are replaced by cm-wlm-setup during installation. If such values are replaced by custom values, then cm-wlm-setup does not change them, and the custom values are used during installation.

If install.config is changed instead of the template file, then cm-wlm-setup replaces it with the configuration file generated from the template. So, changing install.config directly should almost certainly not ever be done.

3. cm-wlm-setup is run. The directory where the LSF files were downloaded is specified on one of the setup screens, or with the --archives-location option.

   Example

   ```
   [root@basecm11 ~]# cm-wlm-setup --wlm lsf --setup --archives-location /root/lsf
   ```

   The same can be achieved by executing cm-wlm-setup without any command line arguments. The required information can be specified within the TUI configuration screens in this case. Also the same can be achieved with the WLM Wizard in Base View.

4. The nodes are then rebooted, and the LSF command bhosts then displays an output similar to:

   Example

   ```
   [root@basecm11 ~]# module load lsf
   [root@basecm11 ~]# bhosts
   HOST_NAME   STATUS  JL/U   MAX  NJOBS   RUN  SSUSP ...
   basecm11        ok     -     2      0     0      0 ...
   head2           ok     -     0      0     0      0 ...
   node001         ok     -     1      0     0      0 ...
   node002         ok     -     1      0     0      0 ...
   node003    unavail     -     1      0     0      0 ...
   node004     closed     -     1      0     0      0 ...
   node005         ok     -     1      0     0      0 ...
   ```

   The output in the preceding example has been truncated for this manual, for convenience.

   The installation status can be checked with the service lsfd (some output elided):

   ```
   [root@basecm11 ~]# systemctl status lsfd
    lsfd.service - IBM Spectrum LSF
      Loaded: loaded (/usr/lib/systemd/system/lsfd.service; enabled; vendor preset: disabled)
      Active: active (running) since Sat 2023-01-14 17:48:05 CET; 1 day 15h ago
     Process: 54334 ExecStart=/cm/shared/apps/lsf/current/etc/lsf_daemons start (code=exited, status=0/SUCCESS)
       Tasks: 13 (limit: 23269)
      Memory: 179.5M
   ```

```
   CGroup: /system.slice/lsfd.service
           |- 54410 /cm/shared/apps/lsf/...10.1/linux2.6-glibc2.3-x86_64/etc/res
           |- 54412 /cm/shared/apps/lsf/...10.1/linux2.6-glibc2.3-x86_64/etc/sbatchd
           |- 54427 /cm/shared/apps/lsf/...10.1/linux2.6-glibc2.3-x86_64/etc/mbatchd -d
                                                          /cm/shared/apps/l...
           |-102312 /cm/shared/apps/lsf/...10.1/linux2.6-glibc2.3-x86_64/etc/eauth -s
           |-344591 /cm/shared/apps/lsf/...10.1/linux2.6-glibc2.3-x86_64/etc/lim -d
   |                                                      /cm/shared/apps/l..
           |-344595 /cm/shared/apps/lsf/...10.1/linux2.6-glibc2.3-x86_64/etc/pim -d
                                                          /cm/shared/apps/...
           |-344612 /cm/shared/apps/lsf/...10.1/linux2.6-glibc2.3-x86_64/etc/melim
           |-344647 /cm/shared/apps/lsf/...10.1_linux2.6-glibc2.3-x86_64/10.1...
           `-457629 /cm/shared/apps/lsf/...10.1_linux2.6-glibc2.3-x86_64/10.1/...

Jan 14 23:29:30 basecm11 mbatchd[54427]: Jan 14 23:29:30 2023 54427:54427 3 10.1 ...
...
[root@basecm11 ~]#
```

while default queues can be seen by running:

```
[root@basecm11 ~]# module load lsf
[root@basecm11 ~]# bqueues
QUEUE_NAME      PRIO STATUS          MAX JL/U JL/P  ...
owners            43  Open:Active       -    -    -  ...
priority          43  Open:Active       -    -    -  ...
night             40  Open:Active       -    -    -  ...
chkpnt_rerun_qu   40  Open:Active       -    -    -  ...
short             35  Open:Active       -    -    -  ...
license           33  Open:Active       -    -    -  ...
normal            30  Open:Active       -    -    -  ...
interactive       30  Open:Active       -    -    -  ...
idle              20  Open:Active       -    -    -  ...
```

The output in the preceding example has been truncated for this manual, for convenience.

If more than one instance of an LSF cluster is set up, then the full modulefile name should be specified. For example:

```
[root@basecm11 ~]# module load lsf/lsf1/10.1
```

Configuring LSF

LSF server configuration: After LSF is set up with `cm-wlm-setup`, the following CMDaemon settings can be modified for the LSF server role:

- `External Server`: a value of `yes` means that LSF server daemons are running on an external server that is not managed by BCM.

The following settings are available in LSF cluster settings, and are applied to all the LSF roles of the LSF cluster:

- `Prefix`: this sets the path to the root location of the LSF installation. The default value is `/cm/shared/apps/lsf/current`.
- `Var`: this sets the path to the `var` directory of LSF. The default value is `/cm/shared/apps/lsf/var`.
- `Cgroups`: this is a submode that contains LSF-related cgroups settings.

The cgroups settings that affect LSF behavior are available using the LSF instance. For example, for an instance `lsf`, the settings can be accessed within `cmsh` using `cmsh->wlm[lsf]->cgroups`. The settings available are:

- `Automount`: if `yes`, then the workload manager tries to mount a subsystem if it is not mounted yet. The default value is `No`.
- `Job Cgroup Template`: this is the template relative job cgroup path. The token `$CLUSTER` specified in this template path is replaced with the actual LSF cluster name, and `$JOBID` is replaced by the job ID. The path is used by the BCM monitoring system in order collect job metrics from the cgroups. By default, the path is set to `"lsf/$CLUSTER/job.$JOBID.*"`.
- `Process Tracking`: if `yes`, then processes are tracked based on job control functions such as: termination, suspension, resume, and other signaling. These are used on Linux systems that support the freezer subsystem under cgroups. The parameter sets LSF_PROCESS_TRACKING in `lsf.conf`.
- `Linux Cgroup Accounting`: if `yes`, then LSF tracks processes based on CPU and memory accounting. This is for Linux systems that support the memory and cpuacct subsystems under cgroups. Once enabled, this parameter takes effect for new jobs. The parameter sets LSF_LINUX_CGROUP_ACCT in `lsf.conf`.

  If this parameter and `Process Tracking` are both enabled, then they take precedence over the parameters LSF_PIM_LINUX_ENHANCE and EGO_PIM_SWAP_REPORT in `lsf.conf`.
- `Mount Point`: specifies a path where cgroups is mounted. It only makes sense to set this when the location is not standard for the operating system.
- `Resource Enforce`: If `yes`, then resource enforcement is carried out through the Linux memory and cpuset subsystems under cgroups. This is for Linux systems with cgroup support. The parameter sets LSB_RESOURCE_ENFORCE in `lsf.conf`.

The server role settings can be modified as follows:

- Within Base View: For example for a head node `basecm11`, using a navigation path of
  `Devices > All Devices > Head Node` > `Settings > Roles` > `+ Add > LsfServerRole`
- Within `cmsh`: For a particular category in the `category` mode, or a particular device in the `device` mode, the `roles` submode is chosen. Within the `roles` submode, the `lsfserver` object can be assigned or used. The following example shows how to set the LSF `prefix` parameter for the `default` category.

  Example

```
[root@basecm11~]# cmsh
[basecm11]% configurationoverlay use lsf-server; roles
[basecm11->configurationoverlay[lsf-server]->roles]% use lsfserver
[basecm11->...[lsfserver]]% set externalserver yes
[basecm11->...[lsfserver*]]% commit
```

The global LSF cluster settings can be modified as follows:

- Within Base View:
  For example, for a head node `basecm11`, using a navigation path of:
  `HPC > Workload Management Clusters > WLM (LSF) instance`
- Within `cmsh`:
  A particular LSF cluster instance can be chosen within `wlm` mode, to set its global parameters. The following example shows how to set the LSF `prefix` parameter for the `lsf1` cluster.

Example

```
[root@basecm11~]# cmsh
[basecm11]% wlm use lsf1
[basecm11->wlm[lsf1]]% set prefix /cm/shared/apps/lsf2
[basecm11->wlm*[lsf1*]]% commit
```

LSF client configuration: After installation, the following CMDaemon settings can be specified for the LSF client role:

- **Queues** In `cmsh`, queues can be added using `jobqueue` mode. In LSF, the default `queue` is `normal`. More arbitrarily-named queues can be specified with:

Example

```
[basecm11->wlm[lsf]->jobqueue]% add abnormal
[basecm11->wlm[lsf]->jobqueue*[abnormal*]]% add subnormal
[basecm11->wlm[lsf]->jobqueue*[subnormal*]]% add supernormal
[basecm11->wlm[lsf]->jobqueue*[supernormal*]]% commit
[basecm11->wlm[lsf]->jobqueue*[supernormal]]% ..
[basecm11->wlm[lsf]->jobqueue*]% list
Name (key)   Nodes
------------ ------------------------
abnormal
normal       node001..node005
subnormal
supernormal
```

The `jobqueue` values can be set for the LSF client role:

Example

```
[basecm11->wlm[lsf]->jobqueue*]% configurationoverlay
[basecm11->configurationoverlay]% use lsf-client
[basecm11->configurationoverlay[lsf-client]]% roles
[basecm11->...->roles]% use lsfclient
[basecm11->...->roles[lsfclient]]% get queues
normal
[basecm11->...->roles[lsfclient]]% set queues abnormal subnormal supernormal normal
[basecm11->...->roles*[lsfclient*]]% commit
[basecm11->...->roles*[lsfclient*]]% show
Parameter                      Value
------------------------------ ------------------------------------------------
Name                           lsfclient
Revision
Type                           LSFClientRole
Add services                   yes
WLM cluster                    lsf
Slots                          auto
All Queues                     no
Queues                         subnormal,supernormal,abnormal,normal
Server                         yes
Host Model
Host Type                      LINUX
GPUs                           0
Node Customizations            <0 in submode>
```

Alternatively, queues can be managed using Base View (section 7.6.2).

- **Specifying nodes for job queues for LSF** The administrator can specify the nodes that are associated with a job queue.

  – Prior to BCM version 11, job queues were specified for nodes only by using the `queues` parameter of the `lsfclient` role at the `device`, `category`, or `configurationoverlay` mode level.

  For example, for the configuration overlay `lsf-client` that defines the LSF client configuration by default, the nodes for the LSF client role are the nodes in the default category `normal`. Within the configuration overlay, within the `lsfclient` role, the default queue to be used is set to `normal` by default:

  Example

```
root@basecm11:~# cmsh
[basecm11]% configurationoverlay use lsf-client
[basecm11->configurationoverlay[lsf-client]]% get categories
default
[basecm11->configurationoverlay[lsf-client]]% roles
[basecm11->configurationoverlay[lsf-client]->roles]% use lsfclient
[basecm11->configurationoverlay[lsf-client]->roles[lsfclient]]% get queues
normal
```

  – Starting with BCM version 11, the administrator can also add nodes to queues directly from within `wlm` mode for a particular WLM, within the `jobqueue` mode for a particular queue, by setting nodes for particular node grouping parameters. For example, for LSF the following queue parameters are available:

  Example

```
[basecm11->wlm[lsf]->jobqueue[normal]]% show | egrep -i '(overlay|nodegroups|compute|catego)'
Overlays
Categories
Nodegroups
Compute nodes
```

  Thus `normal` is used by nodes that have been specified in the following groupings:

  * `Overlays`: nodes in these configuration overlays (including nodes of categories that are in this overlay)
  * `Categories`: nodes in these categories
  * `Nodegroups`: nodes in these node groups
  * `Compute nodes`: nodes listed in this parameter

- **Options** A navigation path of:

  `Devices > All Devices > node001`  `> Settings > Roles > + Add > LSFClientRole`

  lets options be set for a regular node `node001` (figure 7.29):

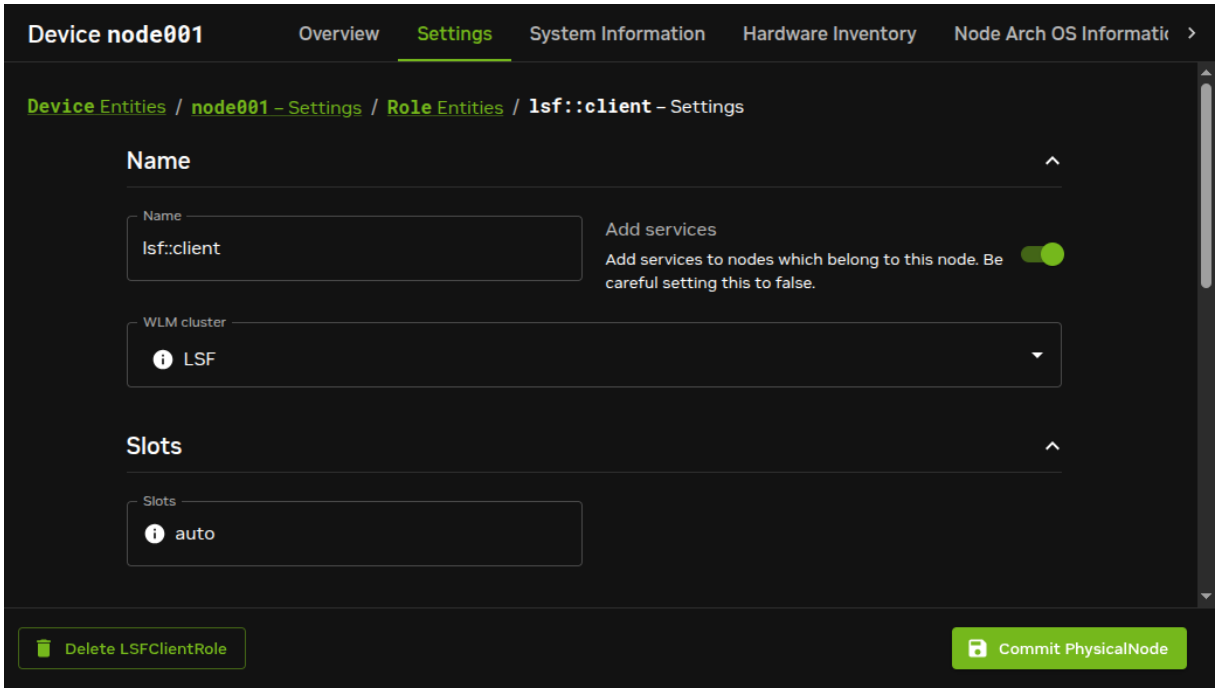


Figure 7.29: Base View access to LSF configuration options using roles window options

Available options are:

- **Slots**: The number of CPUs per node. By default LSF tries to determine the value automatically with the `auto` setting. If the number of slots is set to `0`, then the node becomes an LSF submit host, so that no jobs can run on the host, but users can submit their jobs from this host using the LSF utilities.
- **Queues**: `All` queues can be set, or queues can be selected.
- **Server**: Whether the node is an LSF server
- **Host Model**, **Host Type**: Possible values for these are defined in `lsf.shared`
- **GPUs**: The number of GPUs per node.
- **Node Customizations**: LSF node custom properties (section 7.11).

From `cmsh` these properties are accessible from within the appropriate node or category `roles` submode (section 7.4.2).

Submission hosts: By default, all compute nodes to which the `lsfclient` role is assigned, also become submission hosts. This means that LSF users can submit their jobs from those compute nodes. If the administrator wants to allow users to submit their jobs from a non-compute node—for example from a login node—then the `lsfsubmit` role should be assigned, for example:

Example

```
[root@basecm11~]# cmsh
[...->configurationoverlay[lsf1-submit]]% roles
[...->roles]% assign lsfsubmit
[...->roles*[lsfsubmit*]]% append lsfclusters lsf1
[...->roles*[lsfsubmit*]]% commit
[...->roles[lsfsubmit]]%
```

Configuring a node according to the preceding steps then allows users to submit their jobs from that node, without the submitted jobs getting scheduled to run on that node.

If more than one LSF cluster is set up, and they share the same submit host, then a single `lsfsubmit` role can be used. In this case, all the LSF cluster names should be appended to the `lsfcluster` parameter of the role.

Further configuration: For further configuration the *Administering Platform LSF* manual provided with the LSF software should be consulted.

Running LSF

Role assignment and role removal enables and disables LSF from Base View (sections 7.4.1) or `cmsh` (section 7.4.2).

An active LSF master service (typically, but not necessarily on a head node) has the following LSF-related processes running on it:

Process/Service	Description
`res`	Remote Execution Server*
`sbatchd`	client batch job execution daemon*
`mbatchd`	master batch job execution daemon
`eauth`	External Authentication method
`lim`	Load Information Manager*
`pim`	Process Information Manager*
`pem`	Process Execution Manager*
`vemkd`	Platform LSF Kernel Daemon
`egosc`	Enterprise Grid Orchestrator service controller
`mbschd`	master batch scheduler daemon

*These services/processes run on compute nodes.

Non-active LSF-masters running as compute nodes run the processes marked with an asterisk only.

LSF daemon logs are kept under `/cm/local/apps/lsf/var/log/`, on each of the nodes where LSF services run.

7.6 Using Base View With Workload Management

Viewing the workload manager services from Base View is described in section 7.4.3. The HPC (High Performance Computing) icon, which looks like a speedometer, is the Base View resource that allows the following items to be accessed:

- WLM clusters: for WLM cluster settings for a WLM cluster instance to be viewed and managed
- jobs: for jobs to be viewed and managed
- queues: for queues to be viewed and managed
- job queue stats: for job queue statistics to be viewed

These items are described next.

7.6.1 Jobs Display And Handling In Base View

The navigation path `HPC` > `Jobs` opens up the `Job list` window. This displays a list of recent job IDs, along with the user, status, scheduler used, and queue (figure 7.30).

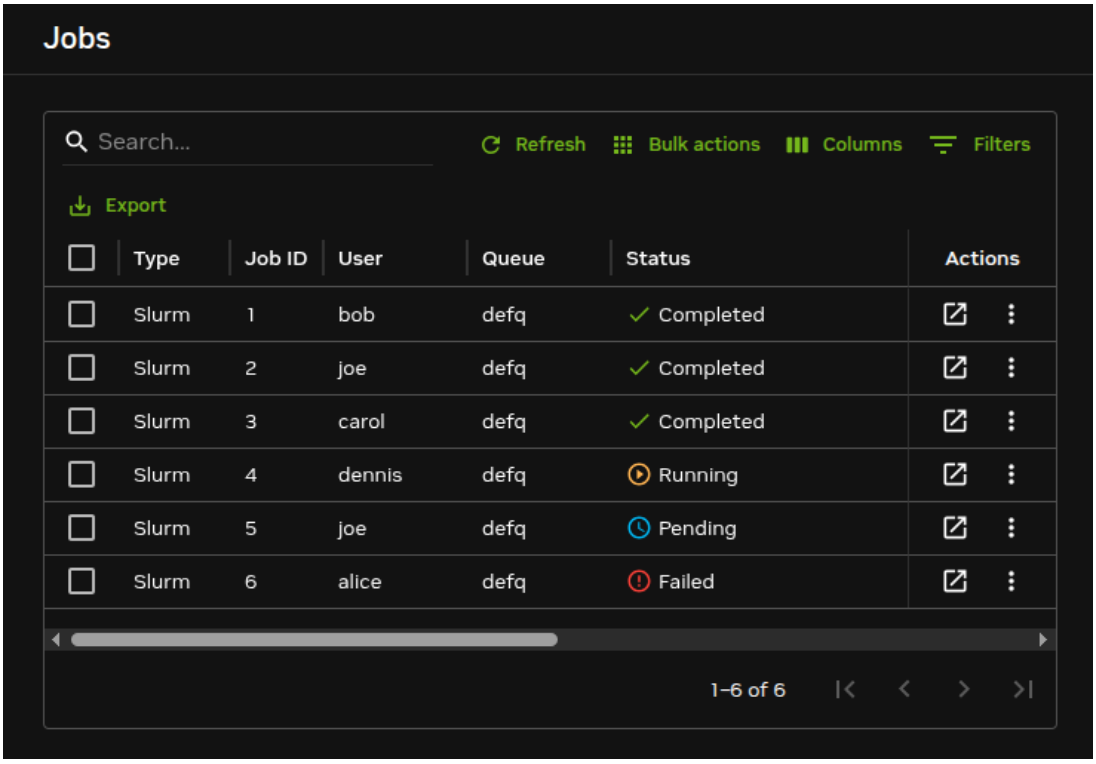


Figure 7.30: Workload manager job list window

- The ⧉ in the `Actions` column opens up a screen that allows a particular job to be examined (figure 7.31):

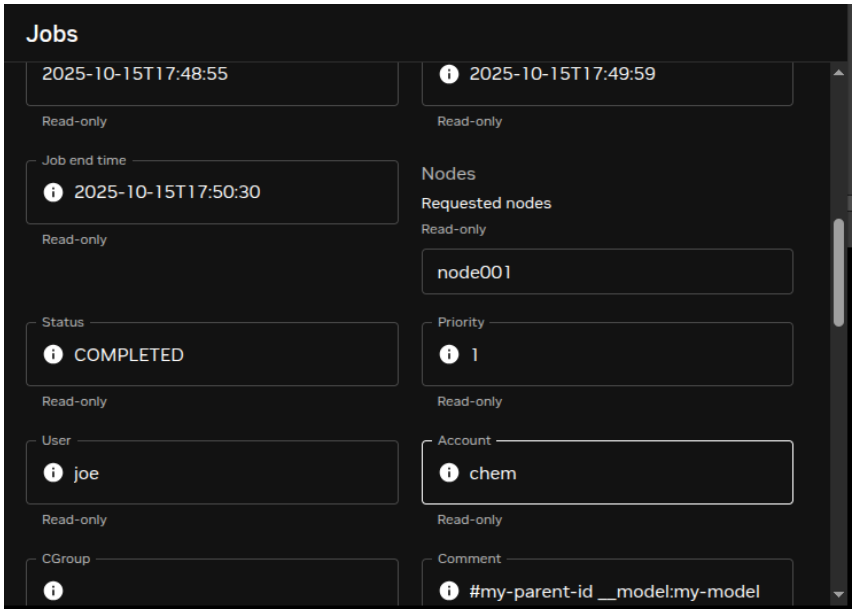


Figure 7.31: Workload manager job subwindow

- The ⋮ in the `Actions` column allows a particular job to be managed (figure 7.32):

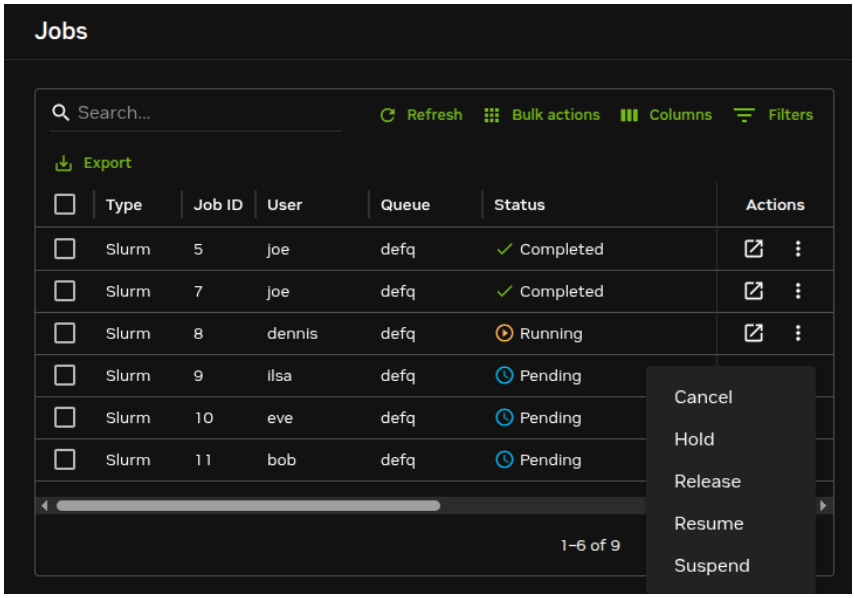


Figure 7.32: Workload manager job management window

The menu options allow the selected job to be managed as follows:

- The `Hold` option stops selected queued jobs from being considered for running by putting them in a `Hold` state (`H` in the `Status` column).
- The `Release` option releases selected queued jobs in the `Hold` state so that they are considered for running again.
- The `Suspend` option suspends selected running jobs (`S` in the `Status` column).
- The `Resume` option allows selected suspended jobs to run again.
- The `Cancel` option removes selected jobs from the queue.

7.6.2 Queues Display And Handling In Base View

The navigation path `HPC > Workload Management Clusters` > `Job Queues` displays a list of queues available (figure 7.33).

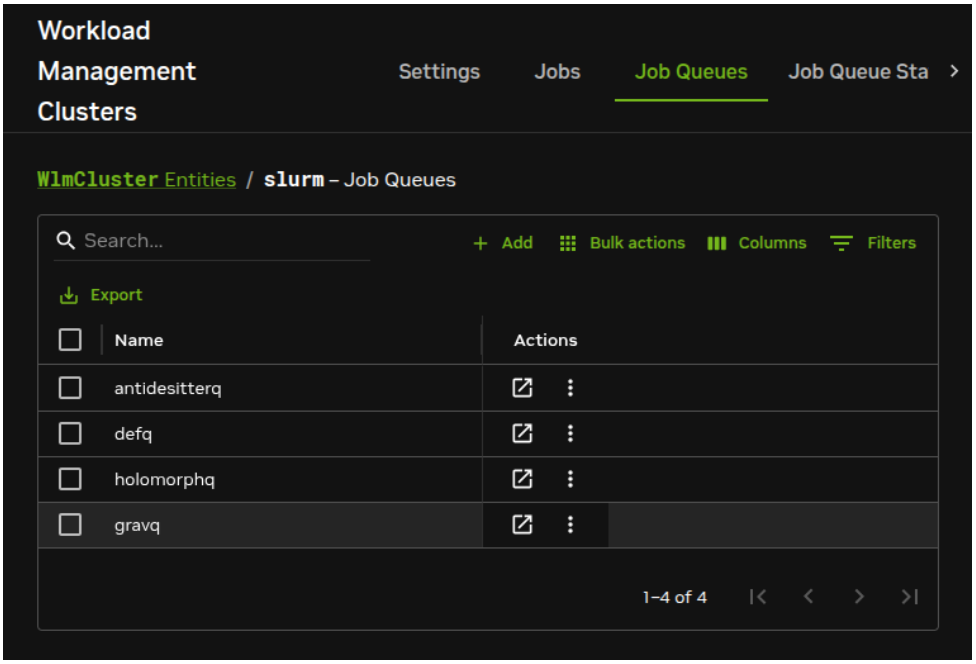


Figure 7.33: Workload manager queues

Queues can be added with the + Add option, or deleted with the ⋮ option. Opening a screen with the ⧉ option allows queue parameters for an existing to be set, for example as in figure 7.34.

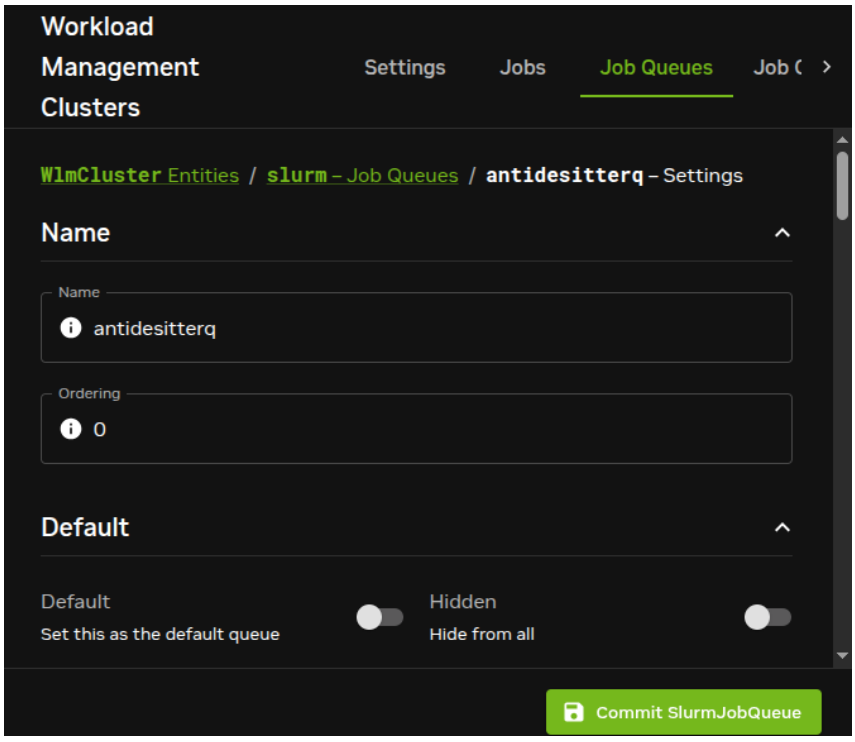


Figure 7.34: Workload manager queue parameters configuration

7.7 Using cmsh With Workload Management

The wlm mode in cmsh gives the administrator access to the WLM instances running on the cluster.

The cmsh tree (Appendix M) has the following submodes structure under the wlm mode:

```
`-- wlm
    |-- accounting
    |-- cgroups
    |-- chargeback
    |-- jobqueue
    |-- jobs
    |-- ocisettings
    `-- placeholders
```

- accounting: provides ways to set some advanced Slurm job accounting properties (page 397)
- cgroups: provides ways to configure WLMs using the cgroup mechanism of the Linux kernel (section 7.10)
- chargeback: provides ways to measure the costs of requested IT resources for WLM jobs (Chapter 13)
- jobqueue: allows WLM job queues to be managed (section 7.7.3)
- jobs: allows WLM jobs to be viewed and managed (section 7.7.2)
- ocisettings: allows OCI settings (Chapter 2 of the *Cloudbursting Manual*) to be viewed and managed
- placeholders: allows nodes to be configured in advance, for planning resource use in WLMs (section 8.4.8)

In addition to these submodes, wlm mode provides several commands at the entity level:

- activeservers: displays the active WLM server nodes (section 7.7.1)
- grid: shows a visual grid of historic job information (section 7.7.2)
- filter: filters job information across WLM instances (page 449)
- statistics: displays job statistics (page 450)

The monitoring commands healthoverview (page 645), samplenow (page 628), and monitoringactionhistory (page 645) are also available in wlm mode, and described in Chapter 10.

For basic WLM management using cmsh, the administrator is expected to use wlm mode with the jobs and jobqueue submodes.

If there is just one WLM instance running on the cluster, then an administrator that accesses wlm mode drops straight into the object representing that WLM instance, and can then access the submodes.

If there is more than one WLM running, then the administrator must select a WLM instance from the top level wlm mode, with the use command, in order to access the submodes.

While at the top level wlm mode, the instances can be listed. The list shows the nodes used by the instance, and the node roles assigned to the nodes.

Suppose BCM is being used to manage two WLM instances:

- one Slurm WLM, named slr
- one Slurm WLM, named uc

with their nodes having been allocated roles as follows:

Example

```
[basecm11->wlm]% list
Type   Name (key)               Server nodes Submit nodes              Client nodes
------ ------------------------ ------------ ------------------------  ----------------
Slurm  slr                      node001      basecm11,node001..node003 node002,node003
Slurm  uc                       node004      basecm11,node004..node006 node005,node006
```

Then one way to list all jobs running per WLM instance, queue, and user, is with the `filter` command:

```
[basecm11->wlm]% filter --running
WLM  Job ID Job name  User  Queue  Submit time Start time   End time  Nodes             Exit code
---- ------ --------- ----- ------ ----------- ------------ --------- ----------------- ----------
slr  103    sjob      maud  defq   15:43:02    15:43:03     N/A       node002,node003   0
slr  104    sjob      maud  defq   15:43:02    15:43:25     N/A       node002,node003   0
slr  105    sjob      maud  defq   15:43:02    15:43:33     N/A       node002,node003   0
slr  106    sjob      maud  defq   15:43:02    15:43:36     N/A       node002,node003   0
slr  171    sjob      maud  defq   14:20:09    14:20:23     N/A       node002,node003   0
uc   200    sjob      fred  batch  14:22:18    14:23:20     N/A       node005,node006   0
```

The `filter` command can also be run within the `jobs` mode, for a particular WLM instance, and is described in more detail on page 449.

An alternate way to list all jobs from `wlm` mode is with a `foreach` command to descend into `jobs` submode to list the jobs for each WLM instance:

Example

```
[basecm11->wlm]% foreach * (jobs ; list)
Type  Job ID  User  Queue  Running time Status     Nodes
----- ------- ----- ------ ------------ ---------- ----------------
Slurm 55      maud  defq   18m 10s      COMPLETED  node002,node003
Slurm 56      maud  defq   18m 5s       COMPLETED  node002,node003
Slurm 57      maud  defq   2m 11s       RUNNING    node002,node003
Slurm 58      maud  defq   0s           PENDING

Type  Job ID  User  Queue  Running time Status     Nodes
----- ------- ----- ------ ------------ ---------- ----------------
Slurm 96      maud  batch  2m 30s       RUNNING    node005,node006
Slurm 97      maud  batch  0s           PENDING
Slurm 98      maud  batch  0s           PENDING
```

The `jobs` submode is now discussed further.

7.7.1 The `activeservers` Command

The `activeservers` command displays the nodes currently acting as active WLM servers. It helps a cluster administrator check which head or server node is actually handling the workload manager, particularly in high-availability configurations.

Example

```
[basecm11->wlm]% activeservers
slurm                    basecm11
```

The output shows the WLM instance name and the hostname of its active server node.

7.7.2 The jobs Submode In cmsh

Within the jobs submode of a WLM instance, the administrator can list jobs that are running or queued up for that particular instance. For example, the running and queued jobs listed for a Slurm instance may be displayed as:

Example

```
[basecm11->wlm[slurm]->jobs]% list
Type       Job ID         User         Queue        Running time Status   Nodes
---------- -------------- ------------ ------------ ------------ -------- -------------
Slurm      3117           user1        hydroq       1s           RUNNING  node002
Slurm      3118           user2        workq        1s           RUNNING  node001
Slurm      3119           user1        hydroq       0s           PENDING
Slurm      3120           user3        hydroq       0s           PENDING
Slurm      3121           user1        hydroq       0s           PENDING
Slurm      3122           user1        hydroq       0s           PENDING
```

The number of jobs can be very large. The wait command in jobs mode can be used to provide a shorter running summary. The following wait command displays a running summary that regularly displays a tally of the running and pending jobs; waits for all of them to complete; and has the waiting-for-completion condition overruled by a timeout of 10 minutes:

Example

```
[basecm11->wlm[slurm]->jobs]% wait --timeout 10m --all
          0s| running: 4, pending: 12
          1m| running: 3, pending: 12
          2m| running: 4, pending: 11
          5m| running: 4, pending: 10
          7m| running: 4, pending: 9
          9m| running: 3, pending: 8
* timeout *
```

By default wait runs every minute until the specified timeout. Also, by default, as indicated by the missing 3rd and 4th minutes in the example, the outputs are only displayed with the tally changes. Further options to the wait command within jobs mode can be found in the help text display output when help wait is run.

In the foreach jobs listings for the two Slurm WLM instances shown earlier, the jobs are in a running, queued up, or completed state. They are also all being run by a user maud, who happens to be making use of both workload managers.

For Slurm, completed jobs are shown for a short time with the status COMPLETED.

Commands That Change The Status Of A Queued Or Running Job

Within a jobs submode, the following commands, used with a job ID, can change the status of a particular queued or running job:

- hold: puts a queued job into a hold state. This prevents the job from being considered for running.
- release: releases a job from a hold state, putting it back in the queue, so that it can be considered for running.
- suspend: pauses a running job
- resume: resumes a suspended job
- remove: removes a job

For example, continuing the case earlier on of the `cmsh` session with the `foreach` listing (page 444):

The administrator can `suspend` the running job with job ID 57 on the Slurm instance as follows:

Example

```
[basecm11->wlm]% jobs slr; suspend 57
[basecm11->wlm[slr]->jobs]% list | head -2; list | grep 57
Type  Job ID  User  Queue  Running time Status     Nodes
----- ------- ----- ------ ------------ ---------- ----------------
Slurm 57      maud  defq   2m 51s       SUSPENDED  node002,node003
```

The administrator can then `resume` the suspended job ID 57 as follows:

Example

```
[basecm11->wlm[slr]->jobs]% resume 57
```

Commands To Inspect Jobs:
Within the `jobs` submode, the following commands can be used to inspect a job:

- `show` <*job ID*>: properties are shown for a running or pending job.

  The properties that are displayed are taken directly from commands associated with particular workload managers:

  - For Slurm: `scontrol show jobs`
  - For LSF: `bjobs`

- `info` <*job ID*>: details are shown about a particular job ID. The job can be `pending`, `running`, or `completed` jobs. The data is taken from the BCM database.

- `statistics`: statistics are shown for jobs. More detail on the `statistics` command is given on page 450.

- `list`: lists pending and running job IDs along with some other associated properties. Recently completed jobs are also displayed. Completed jobs do not remain listed for longer than a few minutes. The `filter` command (section 7.7.2) can be used instead to display all jobs for that WLM instance, using data taken from the BCM database.

- `dumpmonitoringdata` [*OPTIONS*] [<*start-time*> <*end-time*>] <*metric*> <*job id*>: Displays the monitoring data for a specific metric or healthcheck for a running or completed job. Example: `dumpmonitoringdata CtxtSwitches 170`

  More detail is given on the `dumpmonitoringdata` command in section 10.6.4.

- `latestmonitoringdata` [*OPTIONS*]<*job ID*>: Displays the last measured job monitoring data for the job.

  - `latesthealthdata` <*job ID*>: The subset of health checks within the last measured monitoring data.
  - `latestmetricdata` <*job ID*>: The subset of metrics within the last measured monitoring data.

  More detail is given on the `latest*data` commands in section 10.6.3.

- `measurables` <*job ID*>: The measurables used for job monitoring data.

- healthchecks <*job ID*>: The subset of health checks within the measurables used for job monitoring data.
- metrics <*job ID*>: The subset of metrics within the measurables used for job monitoring data.
- enummetrics <*job ID*>: The subset of enummetrics within the measurables used for job monitoring data.

The info Command Within jobs Submode

Within the jobs submode of wlm, the info <*job ID*> command shows job information details stored in the BCM database. BCM starts updating the job information in its database as soon as it detects a new job. When the job is completed BCM pulls a final job state and exit code from the workload manager.

Example

```
[basecm11->wlm[slr]->jobs]% info 3
Parameter                      Value
------------------------------ ------------------------------------------------------
Job ID                         3
Revision
Job name                       matrix-math---model-other-model-args:3-tasks:1.sbatch
Job array ID
Job task ID
User                           joe
Group                          joe
Account                        math
Accounting info                < not set >
Workload manager cluster       slurm
Queue                          defq
Nodes                          node002
Submit time                    27/07/2026 02:35:24
Start time                     27/07/2026 02:35:24
End time                       27/07/2026 02:36:25
Run time                       1m 1s
Persistent                     no
Exit code                      0
Status                         COMPLETED
Requested CPUs                 1
Requested CPU cores            0
Requested GPU                  0
Requested memory               3.2GiB
Requested slots                0
Monitoring                     yes
Comment                        __model:other-model
Run directory                  /home/joe/war
Stdin file                     /dev/null
Stdout file                    /home/joe/war/matrix-3.out
Stderr file                    /home/joe/war/matrix-3.err
Total power usage              < not set >
Total GPU power usage          < not set >
Total CPU power usage          < not set >
Total power under allocation   < not set >
QOS                            normal
[basecm11->wlm[slurm]->jobs]%
```

The power usage fields for WLM jobs in the preceding output: Total power usage, Total GPU power usage, Total CPU power usage, and Total power under allocation, are populated based on

metrics configured in the `wlmjobpowerusagesettings` submode in `partition` mode (described later on in this section).

For the associated metrics, power usage is calculated per job, for completed jobs by computing the energy consumption (the sum of power consumed over time) from the job start time until the job end time.

If no metrics are configured, then the power usage fields remain `< not set >`.

When the metrics are configured, the `info` output for a completed job shows its energy consumption:

Example

```
Total power usage               0 Wh
Total GPU power usage           556.649 Wh
Total CPU power usage           61.2204 Wh
Total power under allocation    0.0%
```

The change from `< not set >` to values only show up some time after the job has completed, to allow values to be properly gathered.

The `wlmjobpowerusagesettings` submode under `partition` mode: has power usage parameters that can be associated with metrics:

Example

```
[basecm11->partition[base]->wlmjobpowerusagesettings]% show
Job age                         5m
Timeout                         30s
Disabled                        no
Node power metrics
GPU power metrics
CPU power metrics
Power under allocation metrics
```

- `Job age`: Minimum age of a job before power usage is calculated.
- `Timeout`: Timeout for power data collection.
- `Disabled`: Disables job power usage calculation when set to `yes`.
- `Node power metrics`: Metrics used for total node power usage.
- `GPU power metrics`: Metrics used for GPU power usage.
- `CPU power metrics`: Metrics used for CPU power usage.
- `Power under allocation metrics`: Metrics used for power consumption under allocation.

Metrics can be configured to be part of the power metrics by adding or appending them:

Example

```
[basecm11->partition*[base*]->wlmjobpowerusagesettings*]% set nodepowermetrics<tab><tab>
Display all 350 possibilities? (y or n)
alertlevel:count            managedservicesok:pass       sys_class_net_addr_assign_type:eth1
alertlevel:maximum          managedservicesok:total      sys_class_net_carrier:ens3
alertlevel:sum              managedservicesok:unknown    sys_class_net_carrier:eth0
...
[basecm11->partition*[base*]->wlmjobpowerusagesettings*]% set nodepowermetrics utilization:cmd
...
```

```
[basecm11->partition*[base*]->wlmjobpowerusagesettings*]% append nodepowermetrics udpinerrors
[basecm11->partition*[base*]->wlmjobpowerusagesettings*]% show
Parameter                      Value
------------------------------ ------------------------------------------------
Job age                        5m
Revision
Timeout                        30s
Disabled                       no
Node power metrics             UdpInErrors,Utilization:cmd
...
```

The filter Command Within jobs Submode

If run at the top-level wlm mode, then the filter command lists jobs for all WLM instances.

It can also be run at the submode-level jobs mode, for a particular WLM instance, in which case it lists only the jobs for that instance (section 11.5).

The filter command without any options is a bit like an extension of the list command in that it lists currently pending and running jobs, although in a different format. However, in addition, it also lists past tasks, with their start and end times.

An example to illustrate the output format for a Slurm instance, somewhat simplified for clarity, is:

Example

```
[basecm11->wlm[myslurminstance]]% filter
Job ID Job name       User  Queue Submit time Start time End time Nodes            Exit code
------ -------------- ----- ----- ----------- ---------- -------- ---------------- ---------
2      hello          fred  defq  15:00:51    15:00:51   15:00:52 node001          0
3      mpirun         fred  defq  15:03:17    15:03:17   15:03:18 node001          0
...
25     slurmhello.sh  fred  defq  16:17:30    16:17:31   16:17:33 node001..node004 0
26     slurmhello.sh  fred  defq  16:18:39    16:18:40   16:19:02 node001..node004 0
27     slurmhello.sh  fred  defq  16:18:41    16:19:03   16:19:25 node001..node004 0
28     slurmhello.sh  fred  defq  16:18:41    N/A        N/A      node001..node004 0
29     slurmhello.sh  fred  defq  16:18:42    N/A        N/A      node001..node004 0
```

In the preceding example, the start and end times for jobs 28 and 29 are not yet available because the jobs are still pending.

Because all jobs for that WLM instance—historic and present—are displayed by the filter command, it means that finding a particular job in the output displayed can be hard. The displayed output can therefore be filtered further.

The following options to the filter command are therefore available in order to find particular groupings of jobs:

- -w|--wlm: by WLM instance
- -u|--user: by user
- -g|--group: by group
- --ended: by jobs that ended successfully, with a zero exit status
- --running: by jobs that are running
- --pending: by jobs that are pending
- --failed: by jobs that finished unsuccessfully, with a non-zero exit status

It may take about a minute for CMDaemon to become aware of job data. This means that, for example, a job submitted 10s ago may not show up in the output.

Further options to filter command can be seen by running the command help filter within the wlm or jobs modes.

The `statistics` Command

The `statistics` command without any options displays an overview of states for all past and present jobs:

Example

```
[basecm11->wlm[uc]]% statistics
Pending    Running    Finished   Error      Nodes
---------- ---------- ---------- ---------- ----------
4          0          154        5          309
```

The options available include the following:

- `-w|--wlm` <*WLM instance*>: by WLM instance
- `-u|--user`: by user
- `-g|--group`: by group
- `-a|--account`: by account
- `-i|--accounting-info <field>[,<field>,...]`: by free text accounting fields
- `--hour` : by hour
- `--day`: by day
- `--week`: by week
- `--interval`: by interval

The jobs statistics can be split across users. An example of the output format in this case, somewhat simplified for clarity, is:

Example

```
[basecm11->wlm[slurm]->jobs]% statistics --user
User        Pending    Running    Finished   Error      Nodes
----------------------- ---------- ---------- ---------- ----------
alice       0          0          2          0          1
bob         0          0          5          0          4
charline    0          1          2          0          2
dennis      0          0          6          0          4
eve         0          0          4          0          1
frank       0          0          1          0          1
```

The job statistics can be displayed over various time periods, if there are jobs within the associated period. An example of the output format for an interval of 60s is:

Example

```
[basecm11->wlm[slurm]->jobs]% statistics --interval 60
Date                Pending    Running    Finished   Error      Nodes
------------------- ---------- ---------- ---------- ---------- ----------
2020/05/13 15:00:00 0          0          1          0          1
2020/05/13 15:03:00 0          0          7          0          7
2020/05/13 15:04:00 0          0          1          1          2
2020/05/13 15:06:00 0          0          3          0          7
2020/05/13 15:07:00 0          0          1          0          0
2020/05/13 15:08:00 0          0          6          0          8
```

```
2020/05/13 16:17:00  0          0          1          0          4
2020/05/13 16:19:00  0          0          9          0          36
2020/05/13 16:20:00  0          0          6          0          24
2020/05/13 16:21:00  0          0          3          0          12
```

In the preceding example there are no jobs within the period associated with the time 15:01 and 15:02. For readability the statistics for those intervals are not displayed.

The grid Command

The grid command displays a color-coded visual grid of historic job information in the terminal. Its use is described in section 12.8.

Example

The following shows job allocation by user for the last hour, in 5-minute intervals, with values displayed as text:

```
[basecm11->wlm[slurm]]% grid --start -1h --end now --interval 5m --mode user --text
```

The output is a color-coded terminal grid showing which users ran jobs on each node during each time interval, along with utilization percentages.

Additional options allow filtering by node, category, rack, or job attributes:

Example

```
[basecm11->wlm]% grid --start -30d --end now --interval 6h --mode account --text --user alice,bob
```

The help text for the grid command (help grid) provides further details on all available options.

The trackedjobs Command Within jobs Submode

The trackedjobs command shows how many jobs are currently being tracked per node by CMDaemon:

Example

```
[basecm11->wlm[slurm]->jobs]% trackedjobs
Node     Jobs
-------- --------
node001  15
node002  15
node003  15
node004  16
```

This is useful for verifying that CMDaemon is aware of job activity on each node.

The pidsgpus Command Within jobs Submode

The pidsgpus command shows the process IDs (PIDs) and GPU indices detected on each node that is running a particular job. This is useful for inspecting which processes a job has spawned and which GPUs they are using.

Example

```
[basecm11->wlm[slurm]->jobs]% pidsgpus 10
node001  0, 1, 2, 3  7205 7222 7225 7226 7227 7228 7229 7230 7231 7232 7233
```

Each line shows the node name, the GPU indices assigned to the job, and the PIDs of the job's processes on that node. For multi-node jobs, one line per node is displayed.

If the job has completed and is no longer tracked, the output indicates this:

Example

```
[basecm11->wlm[slurm]->jobs]% pidsgpus 3
node003  Job not tracked
node004  Job not tracked
```

The stdout, stderr, And stdin Commands Within jobs Submode

The stdout, stderr, and stdin commands display the standard output, standard error, and standard input of a job respectively:

Example

```
[basecm11->wlm[slurm]->jobs]% stdout 10
[Sat Feb 28 13:16:41 2026] start, process 0 of 2 on node001
[Sat Feb 28 13:16:41 2026] sleep, process 0 of 2 for 1 (9)
[Sat Feb 28 13:16:41 2026] start, process 1 of 2 on node001
[Sat Feb 28 13:16:41 2026] sleep, process 1 of 2 for 1 (9)
...
```

These commands retrieve the output files from the node where the job ran. The file paths for stdout, stderr, and stdin are shown in the info output for the job (the Stdout file, Stderr file, and Stdin file fields).

The persistent And forget Commands Within jobs Submode

The persistent command marks a job so that it is not automatically removed from the job information cache. This is useful for keeping a record of important jobs beyond the normal retention period configured by JobInformationKeepDuration (page 935) and JobInformationKeepCount (page 935):

Example

```
[basecm11->wlm[slurm]->jobs]% persistent 10
```

The persistent status of a job is shown in the info output as the Persistent field.

The forget command is the opposite: it removes a job from the cache immediately, regardless of whether it is marked as persistent:

Example

```
[basecm11->wlm[slurm]->jobs]% forget 1
```

Job Directives

Job directives configure some of the ways in which CMDaemon manages job information processing.

The administrator can configure the following directives:

- JobInformationKeepCount: The maximal number of jobs that are kept in the cache, default 8192, maximal value 65536 (page 935).
- JobInformationKeepDuration: How long to keep jobs in the CMDaemon database, default 28 days (page 935).
- JobInformationMinimalJobDuration: Minimal duration for jobs to place them in the cache, default 0s (page 936).
- JobInformationFlushInterval: Over what time period to flush the cache to storage (page 936).
- JobInformationDisabled: Disables job information processing (page 934).

7.7.3 Job Queue Display And Handling In cmsh: jobqueue Mode

The jobqueue submode under the top level wlm mode can be used to manage queues for a WLM instance. Queue properties can be viewed and set.

From the top level wlm mode, a list of queues can be seen for each WLM instance by descending into the WLM instance with a foreach:

Example

```
[basecm11->wlm]% foreach * (get name; jobqueue; list)
slr
Name (key)   Nodes
------------ ------------------------
defq         node002,node003
uc
Name (key)   Nodes
------------ ------------------------
batch        node005,node006
[basecm11->wlm]%
```

The usual object manipulation commands (show, get, set and others) of section 2.5.3 work in the jobqueue submode.

Job Queue Parameters For Slurm

For example, for a Slurm instance, the show command might display the following properties:

```
[basecm11->wlm[slr]->jobqueue]% show defq
Parameter                      Value
------------------------------ ------------------------------------------------
All nodes                      node001..node003
Name                           defq
Revision
Type                           Slurm
WlmCluster                     slr
Ordering                       0
Default                        yes
Hidden                         no
Min nodes                      1
Max nodes                      UNLIMITED
Default time                   UNLIMITED
Max time                       UNLIMITED
Priority Job Factor            1
Priority Tier                  1
OverSubscribe                  NO
Alternate
Grace time                     0
Preemption mode                OFF
Require reservation            NO
Select Type Parameters
LLN                            no
TRES Billing Weights
Alloc nodes
CPU bindings                   None
QOS
Default memory per CPU         UNLIMITED
Max memory per CPU             UNLIMITED
Default memory per Node        UNLIMITED
```

```
Max memory per Node             UNLIMITED
Max CPUs per node               UNLIMITED
Default memory per GPU          UNLIMITED
Default CPU per GPU             UNLIMITED
Disable root                    no
Root only                       no
Allow groups                    ALL
Allow accounts                  ALL
Allow QOS                       ALL
Deny Accounts
Deny QOS
ExclusiveUser                   no
Queue State                     None
Nodesets                        ns1
Overlays
Categories
Nodegroups
Compute nodes
Options
```

7.7.4 Nodes Drainage Status And Handling In `cmsh`

Sometimes a node needs to be drained. Draining a node means not allowing new jobs to run on a node, and letting running jobs finish (drain away).

Draining nodes can be for the purpose of:

- planned maintenance
- hardware troubleshooting
- preventing new jobs during system changes
- isolating problematic nodes

Running the `device` mode command `drainstatus` displays if a specified node is in a `Drained` state or not. In a `Drained` state jobs are not allowed to start running on that node.

Running the `device` mode command `drain` puts a specified node in a `Drained` state:

Example

```
[root@basecm11 ~]# cmsh
[basecm11]% device
[basecm11->device]% drainstatus
Node                    Queue                   Status
----------------------- ----------------------- ----------------
node001                 Slurm/slr/defq
node002                 Slurm/uc/batch
[basecm11->device]% drain node001
Node                    Queue                   Status
----------------------- ----------------------- ----------------
node001                 Slurm/slr/defq          Drained
```

The `undrain` command unsets the `Drained` state so that jobs may start running on the node again.

The `drain`, `undrain`, and `drainstatus` commands have the same grouping options. The grouping options can make the command apply to not just one node, but to a list of nodes, a group of nodes, a category of nodes, a rack, a chassis, an overlay, a role, or a status. Continuing the example:

```
[basecm11->device]% drain -c default; !# for a category of nodes
Node                     Queue                    Status
------------------------ ------------------------ ----------------
node001                  Slurm/slr/defq           Drained
node002                  Slurm/uc/batch           Drained
```

The help text for each command indicates the syntax:

Example

```
[root@basecm11 ~]# cmsh -c "device help drain"
Name:  drain - Drain jobs (not data) on a set of nodes

Usage:  drain [OPTIONS/node]

Options:  -n, --nodes <node>
               List of nodes, e.g.
               node001..node015,node020..node028,node030 or
               ^/some/file/containing/hostnames

          -g, --group <group>
               Include all nodes that belong to the node group, e.g.
               testnodes or test01,test03

          -c, --category <category>
               Include all nodes that belong to the category, e.g. default
               or default,gpu

          -r, --rack <rack>
               Include all nodes that are located in the given rack, e.g
               rack01 or rack01..rack04

          -h, --chassis <chassis>
               Include all nodes that are located in the given chassis, e.g
               chassis01 or chassis03..chassis05

          -e, --overlay <overlay>
               Include all nodes that are part of the given overlay, e.g
               overlay1 or overlayA,overlayC

          -m, --image <image>
               Include all nodes that have the given image, e.g default-image
               or default-image,gpu-image

          -t, --type <type>
               Type of devices, e.g node or virtualnode,cloudnode

          -i, --intersection
               Calculate the intersection of the above selections

          -u, --union
               Calculate the union of the above selections

          -l, --role <role>
               Filter all nodes that have the given
               role
```

```
          -s, --status <status>
               Only run command on nodes with specified status, e.g. UP,
               "CLOSED|DOWN", "INST.*"

          --setactions <actions>
               set drain actions, actions already set will be removed
               (comma-separated)

          --appendactions <actions>
               append drain actions to already existing drain actions
               (comma-separated)

          --removeactions <actions>
               remove drain actions

          --listactions
               list all drain actions

          --clearactions
               remove all drain actions


Examples: drain                         Drain the current node
          drain node001                 Drain node001
          drain -r rack01               Drain all nodes in rack01
          drain --setactions reboot     Drain the current node, and reboot when all jobs are completed
```

A useful one-liner to reboot and reprovision a sequence of nodes could be in the following format:

Example

```
cmsh -c 'device; drain -n  <nodes> --setactions powerreset ; drain -n <nodes> --appendactions undrain'
```

This drains the nodes and does a power reset action, which provisions the nodes. After the nodes are up again, they are undrained so that they are ready to accept jobs again. The command allows the sequence of nodes to be rebooted, for example for the purpose of upgrading a kernel, without needing to schedule a downtime for the cluster.

The cluster administrator should be aware that using the preceding one-liner is not a universal solution. For example, it is not a good solution if the drain takes a long time, because it means that nodes that drain early could be idle for a long time.

7.8 Examples Of Workload Management Assignment

7.8.1 Setting Up A New Category And A New Queue For It

Suppose a new node with processor optimized to handle Shor's algorithm is added to a cluster that originally has no such nodes. This merits a new category, `shornodes`, so that administrators can configure more such new nodes efficiently. It also merits a new queue, `shorq`, so that users are aware that they can submit suitably-optimized jobs to this category.

Ways to configure the cluster manager for this are described next.

A New Category And A New Queue With Base View For An Existing Workload Manager Cluster

Queues can be added to a WLM cluster by running the WLM wizard (navigation path `HPC > Wlm Wizard`) if the WLM cluster does not yet exist. The wizard provides an option during WLM cluster creation for adding queues.

To create a new queue in an existing WLM cluster, the navigation path:

`HPC > Workload Management Clusters` > `Job Queues > + Add`

opens up a pop-up menu that allows queues to be added to the WLM cluster. The properties of the new queue can then be set in the dialog window of the queue (figure 7.33)

At least the queue name, `shorq`, must be set for the queue. The configuration for the queue properties is saved, and then the job queue scheduler configuration is saved.

Next, a new category can be added using

`Grouping > Categories > + Add`

Parameters settings in the new category can be set—at least the category name should be set—to suit the new machines The name `shornodes` can therefore be set here and it becomes part of the list of categories (figure 7.35). .

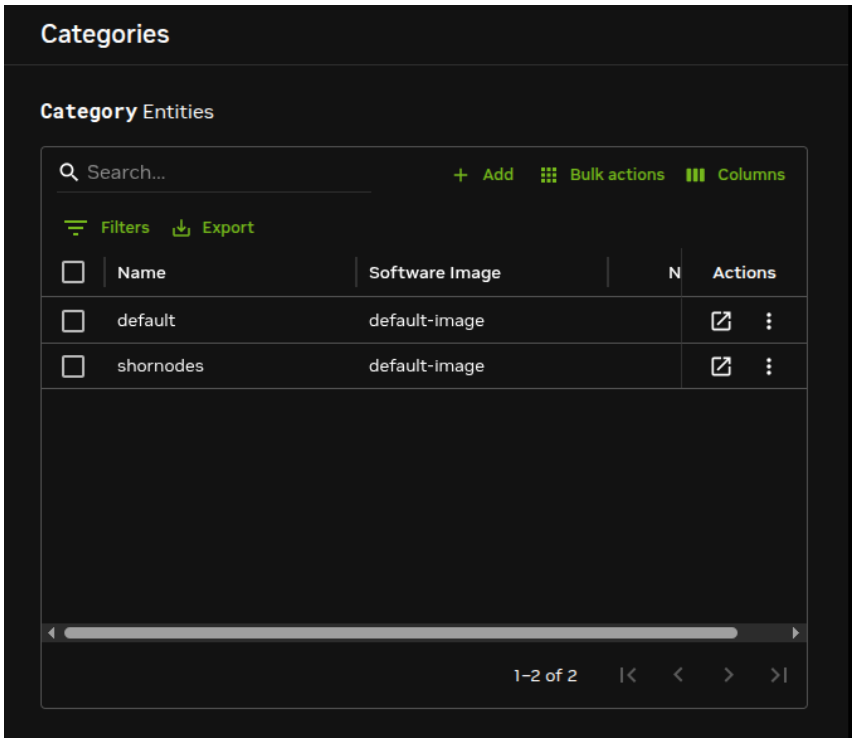


Figure 7.35: A new `shornodes` category created using Base View

Another option within the category is to set a workload manager queue. The queue, `shorq`, is therefore set here for this new category.

Setting a queue for the category means configuring the options of the queue scheduler role, for example the Slurm client role, for the category. So a Slurm client role must be assigned to the category. Continuing from within the `shornodes` category in figure 7.35, the relative navigation path to set a queue is:

> `Settings > Roles` > `+ Add > SlurmClientRole > Queues > shorq`

The appropriate queue can be selected from the queue scheduler menu option (figure 7.36):

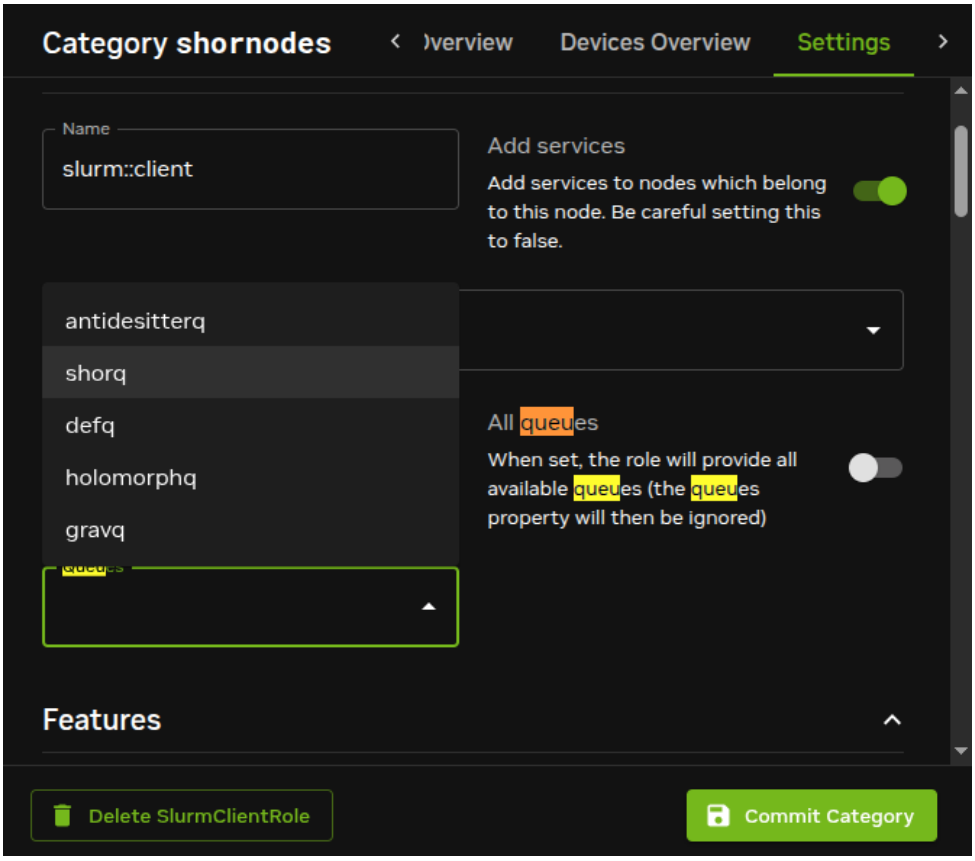


Figure 7.36: Setting a queue for a new category using Base View

In this case, the `shorq` created earlier on is presented for selection. After selection, the category configuration is committed.

Nodes that are to use the queue should be members of the `shornodes` category. The final step is then to allocate such nodes to the category. This can be done, for example for a `node001`, by using the navigation path:

`Devices > All devices > node001` > `Settings > Category`

and setting the category to `shornodes`.

A New Category And A New Queue With `cmsh`

The preceding example can also be configured in `cmsh` as follows:

The new queue can be added from within `jobqueue` mode, for the workload manager. For example, if Slurm is the WLM cluster that is enabled:

```
[basecm11]% wlm; use slurm; jobqueue; add shorq
[basecm11->wlm[slurm]->jobqueue*[shorq*]]% commit
```

The new category, `shornodes`, can be created by cloning an old category, such as `default`:

```
[basecm11->wlm[slurm]->jobqueue[shorq]]% category
[basecm11->category]% clone default shornodes
[basecm11->category*[shornodes*]]% commit
```

Then, going into the roles submode, appropriate workload manager roles can be assigned, and appropriate queues can be appended and committed for that category:

```
[basecm11->category[shornodes]]% roles
```

```
[basecm11->category[shornodes]->roles]% assign slurmclient; commit
[basecm11->category[shornodes]->roles[slurmclient]]% append queues shorq
[basecm11->category[shornodes*]->roles*]% commit
```

The nodes belonging to the shornodes category can then be placed by going into device mode to select the nodes to be placed in that category. For example:

```
[basecm11->category[shornodes]->roles]% device use node002
[basecm11->device[node002]]% set category shornodes
[basecm11->device*[node002*]]% commit
```

7.8.2 Setting Up A Prejob Or Postjob Check

How It Works

Measurables such as health checks (section 10.2.4) by default run as scheduled tasks at timed intervals (page 588). They can independently be configured to run as prejob or postjob checks.

If a health check is configured as a prejob or postjob check, then its response means the same as that of a health check, that is:

- If the response to a prejob or postjob health check is PASS, then it shows that the node is displaying healthy behavior for that particular health check.
- If the response to a prejob or postjob health check is FAIL, then it implies that the node is unhealthy at the time of the check, at least for that particular health check.

Sometimes the strain of running a job can itself cause a health issue on the node. If that is the case, then sometimes the issue can be spotted by an immediate check of the node health after a job has completed—which is what the postjob health check does. A postjob health check is otherwise the same as a regular health check.

- If a postjob health check response is FAIL, then the node is unhealthy, even though the job completed. What effect the unhealthy node had on the job may need consideration by the cluster administrator.
- If a prejob health check response is FAIL, then the node is also unhealthy, but the job is prevented by the WLM from running on that node.

A node that has failed a prejob health check is not allowed to run a job. This is because an unhealthy node means that a job submitted to the node may fail, may not be able to start, or may even vanish outright. The way in which a job can vanish in some cases, without any information beyond the job submission "event horizon" leads to this behavior sometimes being called the *Black Hole Node Syndrome*.

Draining on a prejob or postjob health check failure: It can be troublesome for a system administrator to pinpoint the reason for such job failures, since a node may only fail under certain conditions that are hard to reproduce later on. It is therefore a good policy to disallow passing a job to a node which has just been flagged as unhealthy by a health check. Thus, a sensible action (section 10.2.6) taken by a prejob or postbjob health check on receiving a FAIL response would be to put the node in a Drained state (section 7.7.4). The drain state means that BCM arranges a rescheduling of the job so that the job runs only on nodes that are believed to be healthy.

A node that has been put in a Drained state with a health check is not automatically undrained. The administrator must clear such a state manually.

The failedprejob health check (page 1037) is enabled by default, and logs any prejob health check passes and failures. By default there are no prejob health checks configured, so by default this health check should always pass.

Configuration Using `cmsh`

Prejob and postjob health checks can be enabled for a WLM instance:

```
[basecm11->wlm[slurm]]% show | grep -i job
Enable pre job                  no
Enable post job                 no
[basecm11->wlm[slurm]]% set enableprejob yes; commit
```

Each health check can then be enabled individually within the `monitoring setup` mode.

Example

For example, for the `ib` health check:

```
[basecm11->monitoring->setup]% get ib prejob
no
[basecm11->monitoring->setup]% set ib prejob <TAB><TAB>
yes   no
[basecm11->monitoring->setup]% set ib prejob yes
[basecm11->monitoring->setup*]% commit
================================ ib ======================================
Field                    Message
------------------------ ----------------------------------------------------
when                     warning: Prejob is selected, but no WLM has prejob
```

To allow the healthcheck to run for the prejob with the workload manager, prejob must also be enabled appropriately for the associated workload manager:

Example

```
[basecm11->wlm[slurm]]% get enableprejob
no
[basecm11->wlm[slurm]]% set enableprejob yes
[basecm11->wlm*[slurm*]]% commit
```

Configuration Using Base View

A similar configuration for prejob and postjob checks can be carried out with Base View.

For example, to configure the monitoring of nodes with a prejob check in Base View, the the `Pre job` value can be set for a data producer with the navigation path:

`Monitoring > Data Producers > Monitoring data producers > Edit > Pre job`

Prejob should also be enabled appropriately for the associated workload manager with the navigation path:

`HPC > Workload Management Clusters > Settings > Enable prejob`

Configuration Of Prejob Health Checks With `cm-wlm-setup`

To set a prejob health check up via `cm-wlm-setup`, `cm-wlm-setup` is run in step-by-step mode. During one of the steps the administrator is prompted to select, with a tick on a checkbox, which of the available health checks are to run as prejob health checks in BCM (figure 7.37):

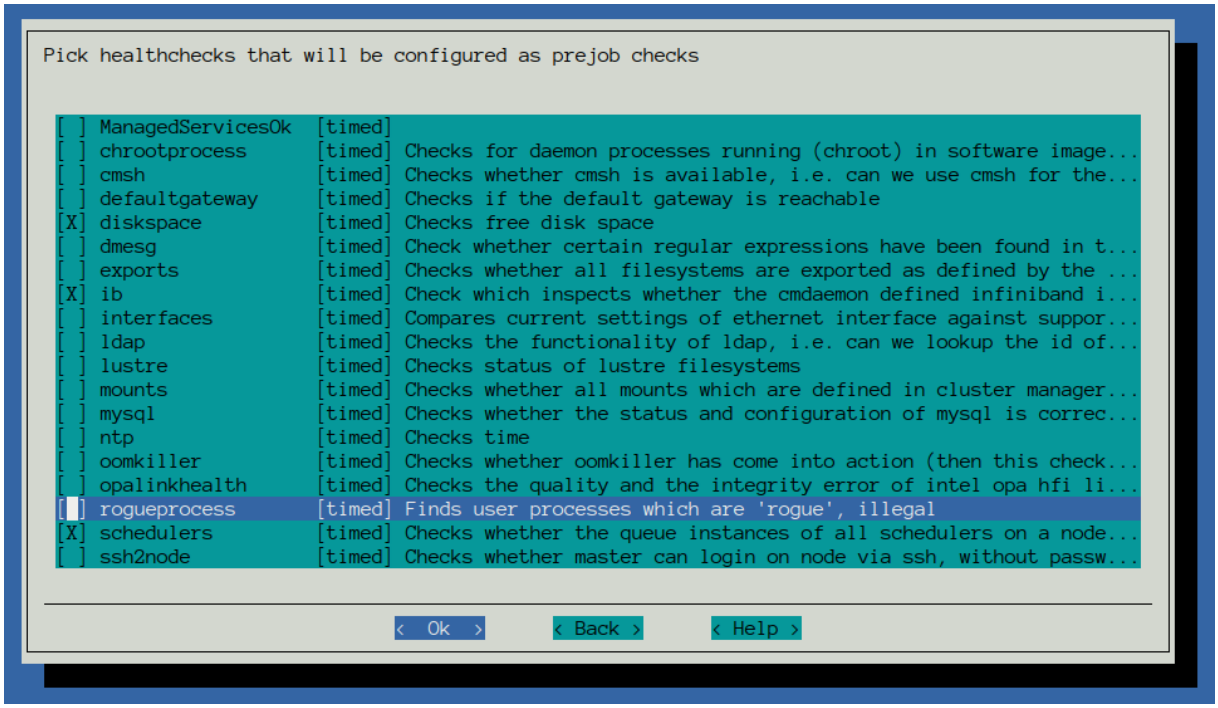


Figure 7.37: Prejob healthcheck selection in the cm-wlm-setup screen

If a selected health check was already set as a prejob—-that is, if the prejob parameter within the monitoring setup mode for that particular health check is already set to yes—then the checkbox is already ticked.

Custom Prejob Configuration

For more unusual prejob or postjob checking requirements, further details on how prologs and epilogs are configured are given in section 7.3.4.

7.9 Power Saving With cm-scale

The cm-scale service can be used by an administrator to reduce the energy and storage costs of compute nodes by changing their power state, or their existence state, according to workload demands. That is, cm-scale automatically scales a cluster up or down, on-demand, by powering up physical nodes, cloud nodes, or virtual nodes. The scaling is carried out according to settings in the ScaleServer role which set the nodes that are to use the cm-scale service.

The cm-scale service is covered extensively in Chapter 8.

7.10 Cgroups

Linux system processes and all their future children can be gathered into sets called process aggregations. These sets can be made into hierarchical groups with specialized behavior using the Control Groups (cgroups) mechanism. The behavior is controlled by different subsystems that are attached to the cgroup. A subsystem may, for example, allow particular CPU cores to be allocated, or it may restrict memory, or it may reduce swap usage by processes that belong to the group, and so on.

Details about Linux cgroups and their subsystems can be found at https://www.kernel.org/doc/Documentation/cgroup-v1/cgroups.txt.

As far as workload management is concerned, it makes sense to distinguish between workload manager cgroup settings and system-wide cgroup parameters. The workload manager cgroup settings allow the administrator to configure a workload manager to use cgroups in a particular way, whereas the

system-wide cgroup settings allow the administrator to manage cgroups whether a workload manager is used or not.

7.10.1 Cgroups Settings For Workload Managers

If the workload manager allows cgroups usage, then BCM provides capabilities to manage the cgroup parameters within the workload manager.

Slurm Cgroups

Slurm supports 3 cgroups-related plugins. These are all enabled by default, and are:

1. `proctrack/cgroup`: enables process tracking and suspend/resume capability using cgroups. This plugin is more reliable for tracking and control than the former `proctrack/linux`.
2. `task/cgroup`: provides the ability to
   - confine jobs and steps to their allocated cpuset
   - bind tasks to sockets, cores and threads
   - confine jobs and steps to specific memory resources and gres devices
3. `jobacct_gather/cgroup`: collects accounting statistics for jobs, steps and tasks using the `cpuacct`, `memory`, and `blkio` cgroups subsystems.

Slurm uses configuration files to store the parameters and devices used for cgroup support.

The file `/cm/shared/apps/slurm/etc/`<*cluster name*>`/cgroup.conf` defines parameters used by Slurm's Linux cgroup-related plugins. The file contains a section that is autogenerated by CMDaemon, with cgroups-related parameters defined in the SlurmServer role.

For Slurm, the administrator can set cgroups parameters using `cmsh` by going into the `cgroups` submode of the WLM instance.

Parameters that can be managed include:

Parameter	Description	Configuration Parameter In `cgroup.conf`
`Auto Mount`*	Force Slurm to mount cgroup subsystems if they are not mounted yet	`CgroupAutomount`
`Mount Point`	Where cgroup root is mounted	`CgroupMountpoint`
`Task Affinity`*	Set a default task affinity to bind each step task to a subset of the allocated cores using `sched_setaffinity`	`TaskAffinity`
`Release Agent Dir`	Directory containing Slurm cgroup release_agent files	`CgroupReleaseAgentDir`

...continues

...continued

Parameter	Description	Configuration Parameter In `cgroup.conf`
`Constrain Cores`*	Constrain allowed cores to the subset of allocated resources	`ConstrainCores`
`Constrain RAM Space`*	Constrain the job's RAM usage	`ConstrainRAMSpace`
`Constrain Swap Space`*	Constrain the job's swap space usage	`ConstrainSwapSpace`
`Constrain Devices`*	Constrain the job's allowed devices (also constrain GPUs) based on GRES allocated resources	`ConstrainDevices`
`Allowed RAM Space`	Percentage memory (default is 100%) out of the allocated RAM allowed for the job cgroup RAM. If this percentage is exceeded, then the job steps will be killed and a warning message will be written to standard error.	`AllowedRAMSpace`
`Allowed Swap Space`	Percent allocated memory allowed for the job cgroup swap space	`AllowedSwapSpace`
`Max RAM Percent`	Maximum percent of total RAM for a job	`MaxRAMPercent`
`Max Swap Percent`	Maximum percent of total RAM for the amount of RAM+Swap that may be used for a job	`MaxSwapPercent`
`Min RAM Space`	Minimum MB for the memory limits defined by `Allowed RAM Space` and `Allowed Swap Space`	`MinRAMSpace`

...continues

...continued

Parameter	Description	Configuration Parameter In `cgroup.conf`

* Boolean (takes `yes` or `no` as a value)

The options are always written in the `cgroup.conf` file. More details on these options can be found in the man page `man cgroup.conf.5`.

LSF

LSF allows resource enforcement to be controlled with the Linux cgroup memory and cpuset subsystems. By default, when LSF is set up with `cm-wlm-setup`, then both subsystems are enabled for LSF jobs. If job processes on a host use more memory than the defined limit, then the job is immediately killed by the Linux cgroup memory subsystem. The cgroups-related configuration options are available in `cmsh` or Base View, and can be found in the `cgroups` submode of the LSF cluster settings:

Parameter	Description	Configuration Parameter In `lsf.conf`
`Resource Enforce`	Controls resource enforcement through the Linux cgroups memory and cpuset subsystem, on Linux systems with cgroups support. The resource can be either `memory` or `cpu`, or both `cpu` and `memory`, in either order (default: `memory cpu`)	`LSB_RESOURCE_ENFORCE`
`Process Tracking`*	This parameter, when enabled, has LSF track processes based on job control functions such as termination, suspension, resume, and other signals, on Linux systems which support the cgroups freezer subsystem	`LSF_PROCESS_TRACKING`

...continues

...continued

Parameter	Description	Configuration Parameter In `lsf.conf`
`Linux Cgroup Accounting`*	When enabled, processes are tracked based on CPU and memory accounting for Linux systems that support cgroup's memory and cpuacct subsystems	`LSF_LINUX_CGROUP_ACCT`

* Boolean (takes `yes` or `no` as a value)

7.11 Custom Node Parameters

NVIDIA Base Command Manager 11 allows the administrator to specify the most important node parameters when a node is configured in a workload manager. But sometimes the workload manager allows the configuration of other advanced or user-defined settings for the nodes. When an administrator clones a node in BCM, or creates it from scratch, then those advanced settings may need to be configured manually per node in the workload manager. Sometimes the nodes are created or cloned automatically, for example `cm-scale` (Chapter 8), in which case node parameters customization is not possible without manual intervention in the workload manager configuration.

Since NVIDIA Base Command Manager version 8.2, the administrator can configure the advanced node settings in the node customizations mode of `cmsh` or Base View. The customizations are available in the workload manager client roles, so they can be applied at the node, node category or configuration overlay levels.

In order to configure a custom node parameter for the workload manager, the administrator adds the node customization entry and sets its value to `True` or `False`. The entry is enabled by default, that is, an associated parameter `Enabled` is `yes` by default. The workload manager can take a minute to implement the new value.

For example, Slurm's `BcastExclude` parameter specifies a list of paths to exclude when broadcasting files to nodes with `sbcast`. It is a `NodeName` parameter in `slurm.conf` but is not directly exposed in BCM's `slurmclient` role, making it a typical candidate for node customizations. To add it as a customization entry, excluding `/dev/shm` from broadcast operations, the following `cmsh` commands can be used:

Example

```
[basecm11]% configurationoverlay
[basecm11->configurationoverlay]% use slurm-client
[basecm11->configurationoverlay[slurm-client]]% roles
[basecm11->configurationoverlay[slurm-client]->roles]% use slurmclient
...->roles[slurmclient]]% nodecustomizations
...->roles[slurmclient]->nodecustomizations]% list
Key (key)    Value        Enabled
------------ ------------ -------
...->roles[slurmclient]->nodecustomizations]% add BcastExclude
...->roles*[slurmclient*]->nodecustomizations*[BcastExclude*]]% set value /dev/shm
...->roles*[slurmclient*]->nodecustomizations*[BcastExclude*]]% show
Parameter                      Value
------------------------------ ------------------------------------------------
Key                            BcastExclude
Value                          /dev/shm
Enabled                        yes
Notes                          <0 bytes>
...->roles*[slurmclient*]->nodecustomizations*[BcastExclude*]]% commit
...->roles[slurmclient]->nodecustomizations[BcastExclude]]% quit
```

The change would show up in slurm.conf within a minute:

```
[root@basecm11 ~]# grep ^NodeName /cm/shared/apps/slurm/etc/slurm/slurm.conf ;    sleep 60 ;    grep ^NodeName /cm/
NodeName=node[001-005] RealMemory=14392 Boards=1 ... Features=location=local
NodeName=node[001-005] BcastExclude=/dev/shm RealMemory=14392 Boards=1 ... Features=location=local
```

Different workload managers allow different kinds of node settings to be set. When the administrator configures node customization entries, BCM configures them differently depending on the workload manager that is used.

Thus if the administrator creates a new customization entry with key name <KEY> and value <VALUE>, then this is applied to the workload managers as follows:

- **Slurm**: The node customization entry is appended to the NodeName line of the particular nodes in slurm.conf in the form <KEY>=<VALUE>. If the entry is removed or disabled in the BCM configuration, then the entry <KEY>=<VALUE> pair is removed from the NodeName line.
- **LSF**: The entry is set in the lsb.hosts file, in the Hosts section. If the <KEY> of the entry is defined as one of the section columns, then the parameter value is replaced by BCM to <VALUE>. When the customization entry is removed or disabled, then its value is replaced by () in the section.

8

NVIDIA Base Command Manager Auto Scaler

8.1 Introduction

NVIDIA Base Command Manager Auto Scaler can be used by an administrator to reduce the energy costs, and the costs associated with (cloud) storage, by compute nodes. This can be done by changing their power state, or their existence state, according to workload demands. The idea behind Auto Scaler is that it automatically scales a cluster up or down, on-demand, by powering up physical nodes or cloud nodes.

On the cluster itself, Auto Scaler is implemented by the `cm-scale` service. The scaling is carried out according to settings in the `ScaleServer` role which set the nodes that are to use the `cm-scale` service.

The `cm-scale` service runs as a daemon. It collects information about workloads from different workload engines, and it uses knowledge of the nodes in the cluster. In the case of HPC jobs, the daemon also gathers knowledge of the queues that the jobs are to be assigned to, and also gathers knowledge on which of the HPC jobs are requesting exclusive node access.

Based on the workload engine information and queues knowledge, the `cm-scale` service can clone and start compute nodes when the workloads are ready to start. The service also stops or terminates compute nodes, when no queued or running workloads remain on the managed nodes.

8.1.1 Use Cases

The `cm-scale` service can be considered as a basis for the construction of different kinds of dynamic data centers. Within such centers, nodes can be automatically re-purposed from one workload engine setup to another, or they can be powered on and off based on the current workload demand.

A few use cases are discussed next, to show how this basis can be built upon:

1. An organization wants to run LSF and Slurm on the same cluster, but how much one workload engine is used relative to the other varies over time. For this case, nodes can be placed in the same pool and shared. When a node finishes an existing job, the `cm-scale` service can then re-purpose that node from one node category to another if needed, pretty much immediately. The re-purposing decision for the node is based on the jobs situation in the LSF and Slurm queues.

2. An organization would like to use their cluster for both Kubernetes and for Slurm jobs. For this case, the admin adds Kubernetes- and Slurm-related settings to the `ScaleServer` role. Using these settings, the `cm-scale` service then switches nodes from one configuration overlay to another. For example, if Slurm jobs are pending and there is a free Kubernetes node, then the node is turned into a Slurm node pretty much immediately. A configuration example for this case is given in section 8.4.9.

3. An organization has a small cluster with Slurm installed on it, and would like to be able to run more jobs on it. This can be carried out using BCM's Cluster Extension cloudbursting capability

(Chapter 3 of the *Cloudbursting Manual*) to extend the cluster when there are too many jobs for the local cluster. The extension to the cluster is made to a public cloud provider, such as AWS or Azure. The `cm-scale` service then tracks the Slurm queues, and decides whether or not new cloud nodes should be added from the cloud as extensions to the local network and queues. When the jobs are finished, then the cloud nodes are terminated automatically. Cluster extension typically takes several minutes from prepared images, and longer if starting up from scratch, but in any case this change takes longer than simple re-purposing does.

4. An organization runs Slurm only and would like to power down free, local, physical nodes automatically, and start up such nodes when new jobs come in, if needed. In this case, `cm-scale` follows the queues and decides to stop or to start the nodes automatically depending on workload demand. Nodes typically power up in several minutes, so this change takes longer than simple re-purposing does.

8.1.2 Resource Constraints

When the service considers whether or not the node is suited for the workload

1. it matches the following requested node resources:
   (a) the number of CPUs (in Kubernetes this value can be fractional);
   (b) the number and type of GPUs (only the number of NVIDIA GPUs is considered for the Kubernetes engine);
   (c) the amount of memory;
2. and, for Kubernetes, the following additional resources:
   (a) pods;
   (b) ephemeral storage;
   (c) extended resources.

   Other types of resources are not considered.

   An extended resource is considered by `cm-scale` if the administrator adds it to the `KUBE_EXTENDED_RESOURCES` list in `config.py`. For example:

Example

```
opts: dict[str, Any] = {
  ...
  "KUBE_EXTENDED_RESOURCES": ["fpga"],
}
```

For HPC workload engines that are unsupported by `cm-scale`, the resources validation is carried out by the HPC engine. If a resource that is requested by the workload cannot be provided by the node, then the HPC engine specifies a pending reason. The pending reason is used by `cm-scale` to decide on node operations.

For Kubernetes, all the used resources must explicitly be specified as extended resources.

When the Kubernetes engine is configured, then `cm-scale` uses the `maxPods` value from the kubelet role as a maximum for the available pod slots per node.

The `parallelism` parameter in the job definition YAML sets the number of pod copies to start. If a Kubernetes job sets the `parallelism` parameter, then `cm-scale` tries to find nodes to satisfy the number of pod copies for the job. The metric `kube_job_spec_parallelism` (page 1025) tracks the `parallelism` value for a job.

For Kubernetes, cm-scale for now supports only NVIDIA GPUs. However, other types of GPUs can be configured in cm-scale as extended resources. If a pod or a job requests a resource with the name nvidia.com/gpu, then cm-scale gets the number of allocatable resources per node in the cluster with this particular resource name, and tries to match it with the pod or job request.

In cmsh, the wlmresources command displays the workload manager engine resources that Auto Scaler considers.

Example

```
[root@basecm11 ~]# cmsh
[basecm11]% device
[basecm11->device]% wlmresources
WLM      Name               Amount          Nodes
-------- ------------------ --------------- ----------------
slurm    cpu_alloc          2               node001..node006
slurm    cpu_total          16              node006
slurm    cpu_total          8               node001..node005
slurm    gpu_free           1               node005
slurm    gpu_free           2               node001..node004
slurm    mem_free           1,000,000,000   node006
slurm    mem_free           380,000,000     node004
slurm    mem_free           6,778,000,000   node005
slurm    mem_free           7,257,000,000   node001..node003
lsf      cpu_total          32              node001..node00+
lsf      gpu_free           1               node005
lsf      gpu_free           2               node001..node00+
lsf      mem_free           0               node001..node003
lsf      mem_free           1,396,000,000   node004
lsf      mem_free           7,456,000,000   node005
lsf      mem_free           990,900,000     node006
lsf      mem_free_per_cpu   1,396,000,000   node004
lsf      mem_free_per_cpu   990,900,000     node006
[basecm11->device]%
```

These resources are taken from the workload managers, and are not necessarily equal to the available physical resources on the nodes.

Number Of CPU Cores Calculation

The cm-scale service uses the number of CPU cores on a node in order to decide on whether a workload should run on the node. The calculation of this number is optimized for many node scenarios.

For each node, at the beginning of each cm-scale iteration, the following procedure is followed step-by-step to determine the CPU cores number. The procedure for the current iteration stops when a step yields a non-zero value, otherwise the next step is tried.

1. If the engine is an HPC workload manager, then the workload manager client role is considered and its slots parameter is used. In the very unusual case of several workload manager roles assigned to the node at the same time, then the minimum number of slots is calculated.

2. If the node is a cloud node, then its flavor Type parameter in the case of EC2, or VMSize in case of Azure, is used. These parameters are accessible via cmsh, within device mode for the cloud node, within the cloudsettings submode. For EC2 nodes the flavor value (Type) can be set to a long statement such as: "62 EC2 Compute Units (16 virtual cores with 3.00 EC2 Compute Units each)", and for this statement 16 is used as the number of CPU cores. If the flavor is not defined on the node, then its cloud provider is considered.

3. If a template node (page 484) is defined in the dynamic node provider, and it is a cloud node, then the template node flavor is used for the CPU core number in the same way as shown in the preceding step.

4. If the `Default Resources` parameter (in a resource provider) contains `cpus:engine=N`, where `N` is a number of CPU cores, then `N` is used. For Kubernetes, the number of CPUs can be fractional.

5. If the node exists at this moment (for example, it is not just cloned in BCM), then the CPU cores number is requested from CMDaemon, which collects nodes system information. This is used as one of the last methods because it is slow.

6. If the node is defined in the `ScaleServer` role via the dynamic resource provider, then the CPU cores number is retrieved from its template node. This method is as slow as the preceding method. It is therefore not recommended when many nodes are managed by `cm-scale`, otherwise the iteration can take several minutes. In such a case, setting the slots parameter manually is typically wiser, so that step 1 is the step that decides the CPU cores number.

If `cm-scale` does not manage to find a value for the node, then it prints a warning in the log file and does not make use of the node during the iteration. It is usually wise for the administrator to set the slots parameter manually for nodes that otherwise persistently stay unused, so that resources are not wasted.

Requested GPUs

When a user of a workload manager requests a number of GPUs for the job that is to be run, then `Auto Scaler` presents nodes that have enough available GPUs for this job. For Slurm, in addition to the number of GPUs, Auto Scaler recognizes the GPU type if the user specifies it. For other workload managers only the number of GPUs is counted.

Auto Scaler knows from CMDaemon via the workload manager what GPUs and what types are available for that workload manager. Auto Scaler tracks the number of GPUs in use, or whether the administrator has configured fewer GPUs than the node actually has. Auto Scaler is therefore aware of what GPUs the workload managers can use during their job scheduling.

If debug messages are enabled in `ScaleServer` role. then the number and type of requested GPUs, as well as the number and types of available GPUs, can be found in the log file at `/var/log/cm-scale.log`

Requested Memory

A memory request is considered by Auto Scaler if a user specifies this request when the job is submitted.

For Kubernetes, if a job or a pod defines a limit, then `cm-scale` uses that limit. If a limit is not set, then Kubernetes limits are followed.

For Slurm, if no memory requirement is specified by the job, then Slurm often sets its own defaults, and `cm-scale` then uses those implicit values.

GE does not provide the amount of memory requested by a user per node, but allocates memory per CPU core. Thus, Auto Scaler by default operates with a memory amount per CPU core, which can also be seen in the logs.

Default Resources Specification

Sometimes the available consumable resources must be defined explicitly by the administrator. This is needed in the case of LSF, because when a node is down, LSF does not provide the available consumable resources configured for the node. Therefore in this case Auto Scaler does not know if the node actually has any of the resources known to LSF. The mechanism used for defining explicitly can also be used for other workload managers, for testing purposes.

For now, only the following types of consumable resources can be specified as default resources to the `Default Resources` setting, under the `Resource Provider` parameter. The setting is a list of strings, where each string specifies one of the following resources:

1. `cpus`: the number of available CPU cores or Kubernetes CPUs, if other sources for this information do not provide a value (the calculation for the number of CPU cores is described on page 469).

   The `cpus` specification has the following format:

   `cpus:`<*engine*>`=`<*amount*>

   where <*engine*> is the name of the workload engine being used and <*amount*> is the amount of available CPUs.

   Example

   ```
   cpus:kube=8.5
   ```

   or

   ```
   cpus:slurm=16
   ```

2. `mem_free`: the amount of available memory. If no units are specified, then bytes are assumed. It is also possible to append one of the following units:

Multiple-byte units			
Decimal SI-style		**Binary IEC-style**	
name	**unit or abbreviation**	**name**	**unit or abbreviation**
kilobytes	KB or K	kibibytes	KiB or Ki
megabytes	MB or M	mebibytes	MiB or Mi
gigabytes	GB or G	gigibytes	GiB or Gi
terabytes	TB or T	tebibytes	TiB or Ti
petabytes	PB or P	pebibytes	PiB or Pi

   The format of the memory specification is the following:

   `mem_free:`<*engine*>`=`<*amount*>

   where <*engine*> is the name of the workload engine which "provides" this value, and <*amount*> is the amount of memory.

   Example

   ```
   mem_free:lsf1=32GB
   ```

3. `gpu_free`: the number of available GPUs. The format of the GPUs specification is:

   `gpu_free[:`<*type*>`]:`<*engine*>`=`<*number*>

   where <*type*> is a string that specifies the GPUs type (available only for Slurm). Only a single GPU type per node is currently supported. <*number*> is a number of GPUs, and <*engine*> is the name of the workload engine which "provides" this value.

   Example

   ```
   gpu_free:a100:slr=8
   ```

   or

   ```
   gpu_free:lsf=1
   ```

8.1.3 Setup

In order to set up Auto Scaler the administrator can run the `cm-auto-scaler-setup` script. The setup allows one of the three base scenarios to be configured. This is possible in express mode as well as in step-by-step mode. When the setup is complete, the administrator can further tune the behaviour of Auto Scaler within the `ScaleServer` role. The `Scaleserver` role is always assigned to the head nodes, via a new configuration overlay named `autoscaler`.

The setup tool assigns the role. The role, in turn, starts the `cm-scale` system service (Auto Scaler). Auto Scaler can be disabled by running `cm-auto-scaler-setup` again, and selecting the menu item `Disable`. When Auto Scaler is disabled, the configuration overlay is removed.

The service writes logs to `/var/log/cm-scale.log`. Running the `cm-scale` daemon in the foreground for debugging or demonstration purposes is also possible, using the `-f` option. Other additional options that may be used, including at the same time, are:

- `-d`: debug logs
- `-i <N>`: number (`<N>`) of iterations

The logs are then duplicated to STDOUT:

Example

```
root@basecm11$ module load cm-scale
root@basecm11$ cm-scale -f
[...]
```

If `cm-auto-scaler-setup` is started on a cluster where Auto Scaler has not been set up, then the administrator can select from one of the following setup operations (figure 8.1):

- express setup,
- step-by-step setup,

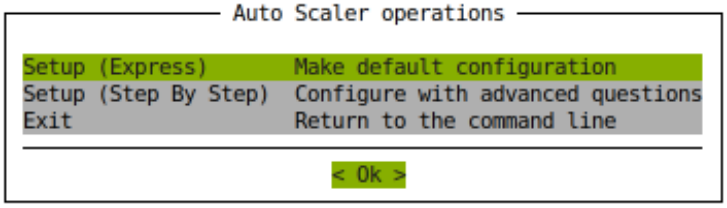


Figure 8.1: Auto Scaler Setup Operations

If Auto Scaler has already been set up, then an option to disable it appears.

- Express setup allows an initial setup to be carried out, without many questions being asked. It applies default values whereever possible. Some inputs that do not have default values are still needed. Express setup is a good start for the administrators who have never used Auto Scaler before.
- Step-by-step setup allows an initial setup to be carried out too, but with some more questions to help tune the cluster to the needs of the administrator. This is more suitable for administrators with some experience in Auto Scaler configuration.

After the wizard has carried out the deployment, both the express and the step-by-step configuration can have their configurations tuned further via `cmsh`, within the `scaleserver` role.

Express setup and step-by-step setup both allow one of the 4 pre-defined use case scenarios to be selected (figure 8.2):



Figure 8.2: Auto Scaler Setup Scenario Selection

These use cases are:

1. **Workload Manager (On-premises)**: Auto Scaler tracks selected workload manager queues, and starts or stops specified on-premises nodes on demand. A static node provider is automatically added to the role for this scenario.

2. **Workload Manager (Cluster Extension)**: Auto Scaler dynamically clones nodes from a cloud template node, on demand. The nodes can be terminated or stopped when idle. This scenario is used when cloudbursting is set up, and there is a cloud director. A dynamic node provider is added automatically to the role for this scenario.

3. **Workload Manager (Cluster in Cloud)**: As in the preceding use case, Auto Scaler dynamically clones nodes from a cloud template node, on demand. In this scenario, there is no cloud director, and the entire cluster must reside in the cloud.

4. **Kubernetes (On-premises)**: Auto Scaler tracks Kubernetes jobs or individual pods, and starts or stops the nodes on demand. A static node provider is automatically added to the role for this scenario.

Static nodes and dynamic nodes providers are discussed in section 8.2.2.

In step-by-step mode, for the next step, the dialog suggests specifying Auto Scaler base options (figure 8.3):

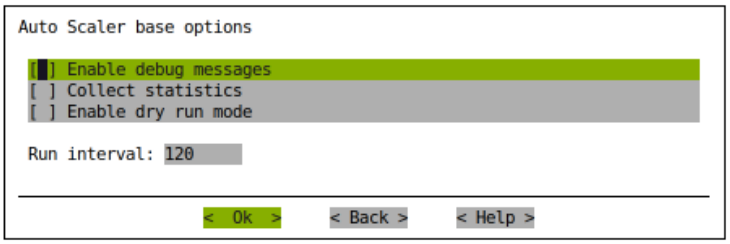


Figure 8.3: Auto Scaler Setup Base Options

The Auto Scaler base options are:

- `Enable debug messages`: Auto Scaler adds debug messages to its log file (default: `/var/log/cm-scale`).
- `Dry run mode`: Disables actual execution of any operation on the cluster. Auto Scaler decisions are still all written to the log file.
- `Run interval`: Number of seconds that Auto Scaler waits before making new decisions regarding cluster auto scaling.

Use Case: Workload Manager (On-premises)

When this scenario is selected, then the next step is to configure the static nodes provider. First, categories and individual nodes to be managed by Auto Scaler are selected. The nodes of a selected category, or individual nodes, are added to the node provider (figures 8.4 and 8.5):

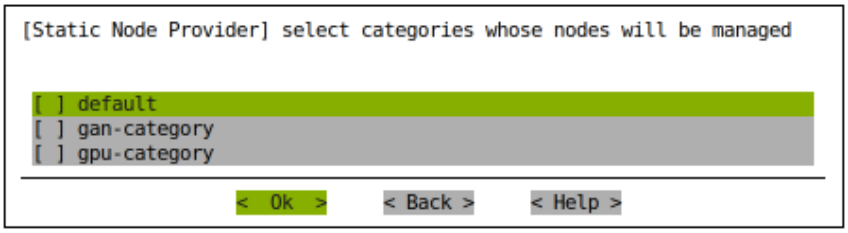


Figure 8.4: Auto Scaler Setup Categories Selection For Static Node Provider

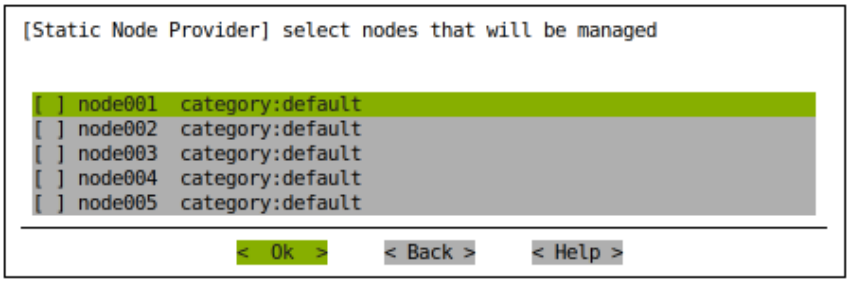


Figure 8.5: Auto Scaler Setup Individual Nodes Selection For Static Node Provider

The next step is to pick the workload manager cluster (figure 8.6):

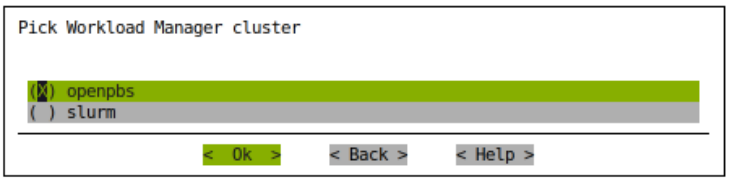


Figure 8.6: Auto Scaler Setup Workload Manager Selection

If only one workload manager instance exists, then the screen is skipped. The screen provides a list of workload manager names that `cm-wlm-setup` has set up. The `cm-auto-scaler-setup` wizard only configures one workload manager cluster, but others can be added later as separate workload engines with the `ScaleServer` role.

If running express mode, then the summary screen is displayed. The summary screen includes options to just show the configuration file, or to save the configuration and deploy the setup (figure 8.7):

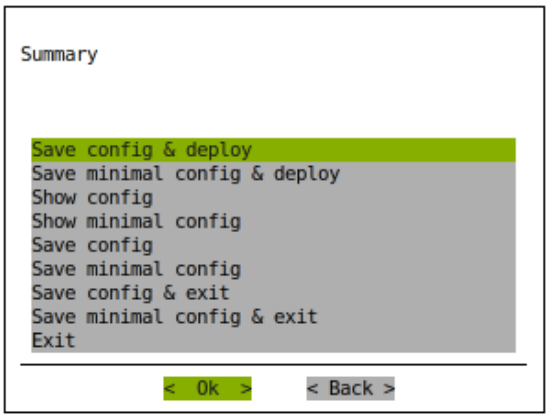


Figure 8.7: Auto Scaler Setup Summary

If running step-by-step mode for the workload manager, then there are some additional tune up screens:

Values can be set for resources in the default node resources screen (figure 8.8):

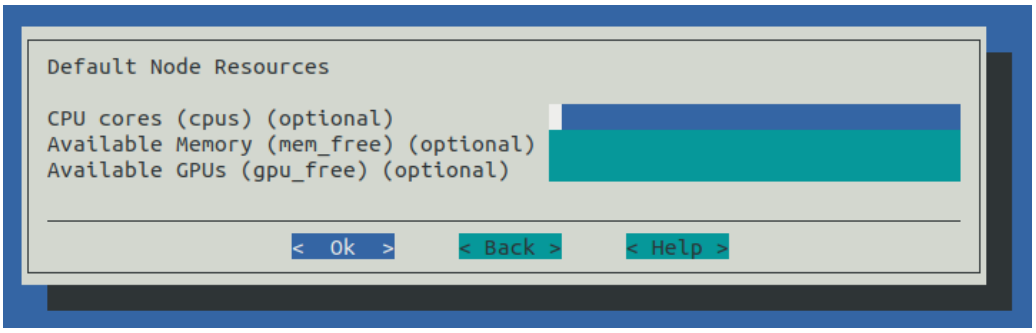


Figure 8.8: Auto Scaler Setup Default Resources

Resources that can be set are:

1. number of CPU cores per node. Format: <*number*>
2. number (and optionally, type) of GPUs per node. Format: [<*type*>]:<*number*>
3. available memory for jobs per node. Units can be specified as: KB, K, KiB, MB, M, MiB, GB, G, GiB, TB, T, TiB, PB, P, PiB. Format: <*amount*>[<*unit*>]. If no units are specified, then bytes are assumed.

When Auto Scaler considers whether or not the node is suited for the workload, it considers the following requested resources:

In `cmsh`, the `wlmresources` command, executed in `devices` mode, displays the resources that Auto Scaler considers. These resources are taken from the corresponding workload manager, and are not necessarily equal to the available physical resources on the nodes. Sometimes the available consumable resources must be defined explicitly by the administrator. This is needed if the node never started, or if the WLM (such as in the case of LSF) does not provide node resource information when the node is down. It is recommended that these values are always defined. If nodes vary in their resource requirements, then, after setup, an administrator can add new resource providers (within the `ScaleServer` role) and set the various default resources for the various groups of nodes.

In the next screen the administrator can specify some WLM engine settings (figure 8.9):

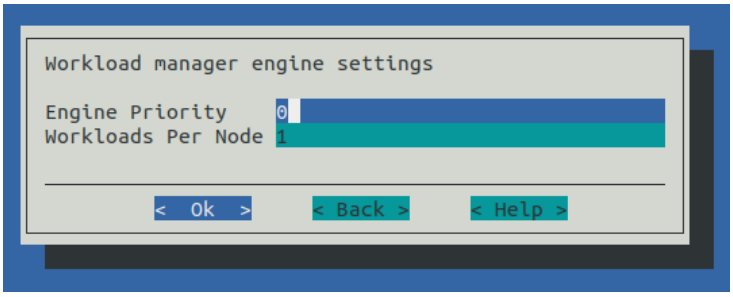


Figure 8.9: Auto Scaler Setup Engine Settings

The settings are:

- `Engine Priority`: The workload engine priority. This value is used when the final (global) workload priority is calculated by Auto Scaler. If `0`, then this priority is not taken into account.
- `Workloads Per Node`: The maximum number of WLM jobs that can be started on a node.

Auto Scaler fetches the workload priority values from the settings specified in the next screen (figure 8.10):

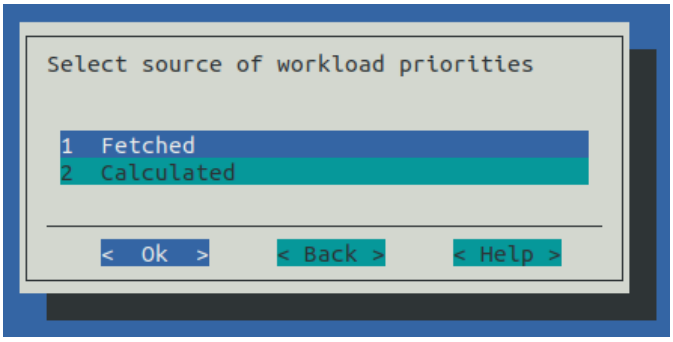


Figure 8.10: Auto Scaler Setup Workload Priorities Source Selection

The settings are:

- `Fetched`: Priorities are fetched from workload engine. Age and engine priorities are ignored. This option sets the age factor to `0.0`, and engine priority to `0`
- `Calculated`: Priorities are calculated from the workload age and engine priority. Both can be tuned by the administrator in `ScaleServer` role. This option sets the age factor to `1.0`, and the external priority factor to `0.0`

Workload trackers settings can be set in the next screen (figure 8.11):

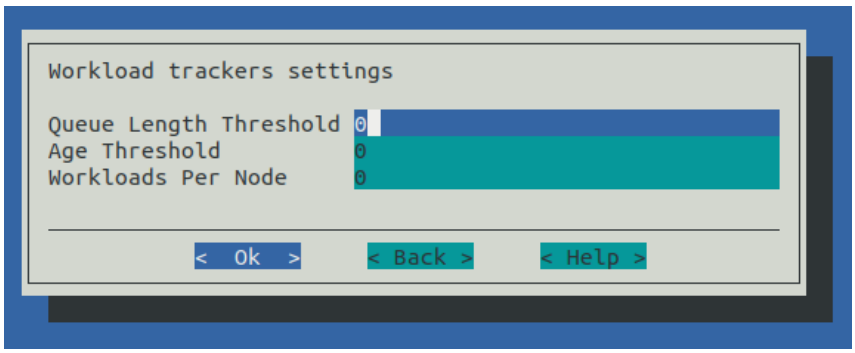


Figure 8.11: Auto Scaler Setup Tracker Settings

The settings are:

- `Queue Length Threshold`: Number of pending workloads. If this number is reached, then nodes are triggered to start up.
- `Age Threshold`: Workload pending time threshold, in seconds. If a workload reaches this age while pending, then nodes are triggerd to start up for that workload.
- `Workloads Per Node`: The maximum number of WLM jobs that can be started on a node for that tracker. A value of `0` means no limit is set.

The settings are applied to all configuring queue trackers. The values can be tuned further afterwards in the `ScaleServer` role.

Use Case: Workload Manager (Cluster Extension)

In the case of workload manager cluster extension scenario, a dynamic node provider with a template node is configured. In this scenario the administrator should expect cloud nodes to be triggered, which run in a previously configured and deployed cluster extension.

Cluster extension configuration and deployment in BCM is described in Chapter 3 of the *Cloudbursting Manual*, and can be carried out, for example, for a particular cloud provider. For command line deployment, the `cm-cluster-extension` setup script can be run.

Cloud nodes are thus cluster nodes that extend into a cloud provider, and which are cloned and terminated depending on workload demand.

To configure a cluster extension with Auto Scaler, the administrator is asked to pick a cloud provider (figure 8.12):

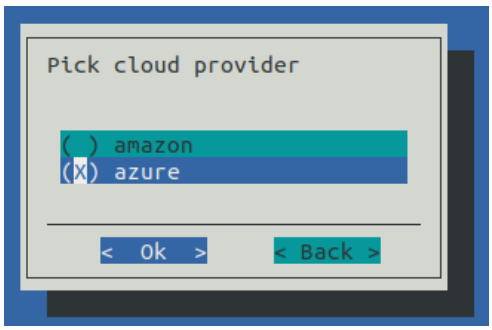


Figure 8.12: Auto Scaler Setup Cloud Provider

With the express setup, the next screen asks for workload manager selection (figure 8.6).

With the step-by-step setup, however, some additional screens are presented before getting to the workload manager selection screen. These extra screens are described next.

The Auto Scaler template node selection screen (figure 8.13) prompts for the selection of a template node that is to be used for cloud node cloning.

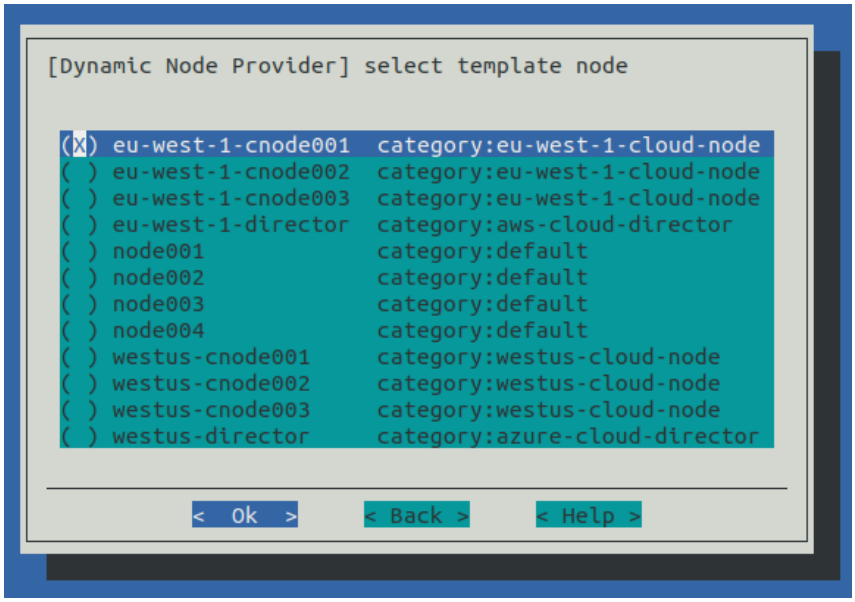


Figure 8.13: Auto Scaler Setup Template Node

The selected template node is then set in the dynamic node provider.

The Auto Scaler incrementing network interface screen (figure 8.14) prompts the administrator to select the network interface on the template node that is automatically incremented when the node is cloned.

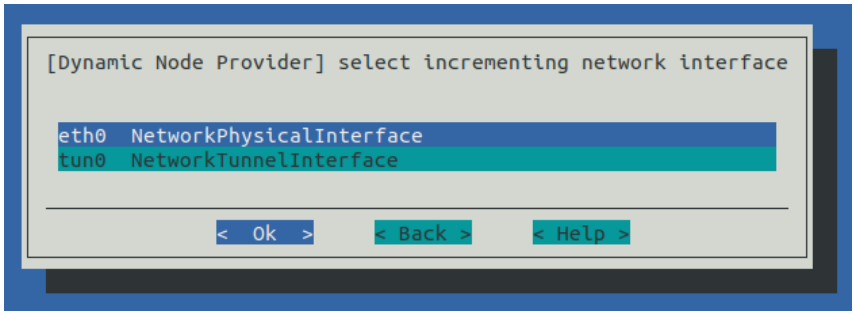


Figure 8.14: Auto Scaler Setup Incremented Network Interface Selection

The Auto Scaler node range specification screen (figure 8.15) prompts the administrator to specify a node range. Range format can be used. Nodes are automatically created by Auto Scaler on demand in the cloud according to the range specified.

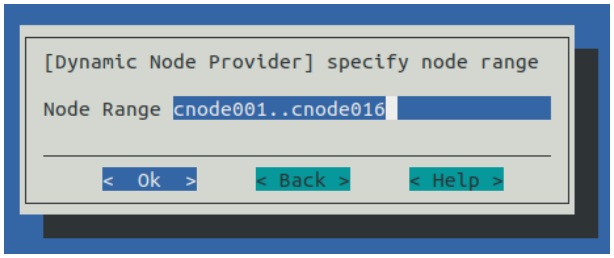


Figure 8.15: Auto Scaler Setup Node Range

The remaining screens in this use case have been covered in the earlier section (**Use Case: Workload Manager (On-premises)**, page 474) , and are:

- default node resources (figure 8.8),
- engine settings (figure 8.9),
- workload priorities (figure 8.10),
- tracker settings (figure 8.11).

Use Case: Workload Manager (Cluster in Cloud)
If a workload manager (Cluster in Cloud) option is chosen from figure 8.2, then the screens that are displayed next follow the same steps as in the preceding case of a workload manager (Cluster Extension) (starting at page 477).

Use Case: Kubernetes (On-premises)
If a Kubernetes (On-premises) scenario is chosen from figure 8.2, and if Kubernetes has been set up (Chapter 4 of the *Containerization Manual*) then the screens that are displayed next are related to Kubernetes and Auto Scaler integration. Auto Scaler tracks Kubernetes jobs or individual pods, and can start or stop the on-premises nodes on demand.

The first screen displayed after the scenario is selected, is a screen that asks for node categories and individual nodes that are to be configured in the static resource provider. Such nodes are the only ones to be managed by Auto Scaler. Category and node selection screens are then displayed as in the earlier sections (figures 8.4 and 8.5).

The administrator is then prompted to select a Kubernetes cluster (figure 8.16).

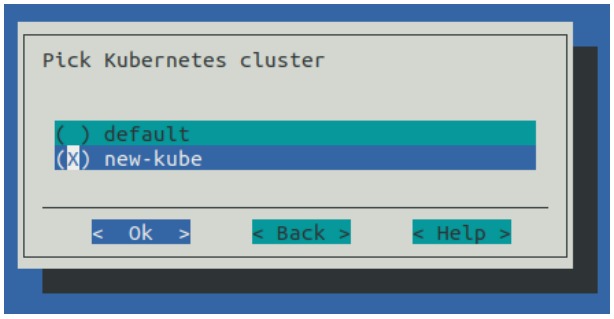


Figure 8.16: Auto Scaler Setup Kubernetes Cluster Selection

If there is only one Kubernetes cluster, then this screen is skipped, and the Kubernetes configuration is used for the integration with Auto Scaler.

The next screen prompts for the Kubernetes engine settings (figure 8.17):

- `Engine Priority`: A workload engine priority. This value is used when the final (global) workload priority is calculated by Auto Scaler. If `0`, then this priority is not taken into account.
- `Workloads Per Node`: The maximum number of Kubernetes jobs, or individual Kubernetes pods without a controller, that can be started on a node.
- `CPU Busy Threshold`: The CPU load % that defines if node is too busy for new pods.
- `Memory Busy Threshold`: The Memory load % that defines if node is too busy for new pods.

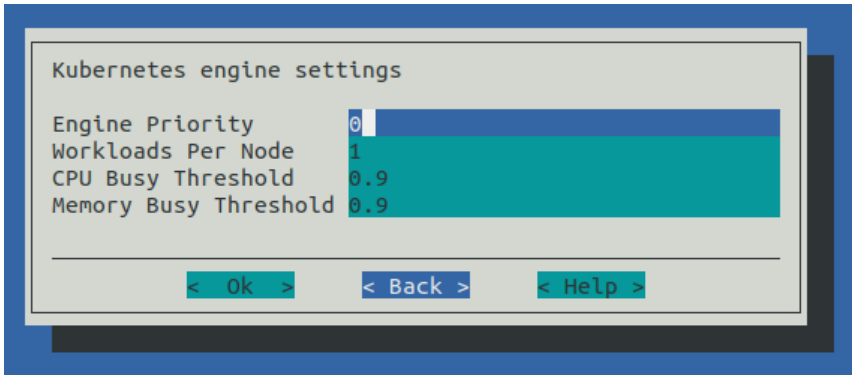


Figure 8.17: Auto Scaler Setup Kubernetes Engine Settings

The administrator is then prompted to pick the Kubernetes namespace that will be tracked by Auto Scaler (figure 8.18):

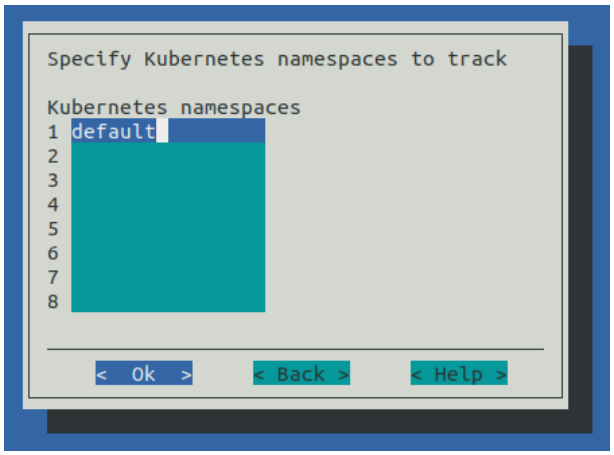


Figure 8.18: Auto Scaler Setup Kubernetes Namespace Selection

In step-by-step mode, the namespace tracker settings are then displayed (figure 8.18):

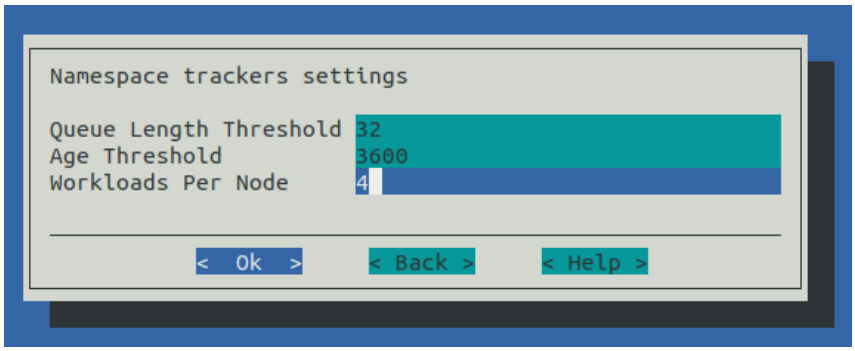


Figure 8.19: Auto Scaler Setup Kubernetes Namespace Tracker Settings

The settings are:

- `Queue Length Threshold`: If this number of pending workloads is exceeded, then that triggers nodes starting up.
- `Age Threshold`: Workload pending time threshold, in seconds. If the age of the workload is greater than this, then nodes are triggered to start for this workload.

- `Workloads Per Node`: The maximum number of Kubernetes jobs, or individual Kubernetes pods without a controller, that can be started on a node for that namespace tracker. A value of `0` means no limit is set.

The settings are applied to all configuring namespace trackers. The values can be tuned further afterwards within the `ScaleServer` role.

The summary screen is displayed next (figure 8.7). Selecting `Save config & deploy` saves the configuration and starts the setup procedure.

8.1.4 Workload Roles Assignment Limitations Per Node With `cm-scale`

The `cm-scale` service allows multiple workload managers to be considered on the same cluster, and besides supporting HPC workload managers, also supports Kubernetes as a type of workload engine.

However, more than one engine role should not be assigned to a node at one time. Thus, for example, assigning a Slurm role and an LSF role at the same time to a node should not be done. Nor, for example, should there be a Kubernetes role and a workload management role assigned at the same time to a node.

8.2 Configuration

8.2.1 The ScaleServer Role

To configure `cm-scale`, the cluster administrator configures the `ScaleServer` role. The role is typically assigned to head nodes:

Example

```
[basecm11]% device use master
[basecm11->device[basecm11]]% roles
[basecm11->device[basecm11]->roles]% assign scaleserver
```

The role is configured by setting values to its settings. There are some advanced settings for less common options:

```
[basecm11->device*[basecm11*]->roles*[scaleserver*]]% show
Parameter                      Value
------------------------------ ------------------------------------------------
Name                           scaleserver
Revision
Type                           ScaleServerRole
Add services                   yes
Engines                        <0 in submode>
Resource Providers             <0 in submode>
Dry Run                        no
Debug                          no
Run Interval                   120
Advanced Settings              <submode>
[basecm11->device*[basecm11*]->roles*[scaleserver*]]% advancedsettings
[basecm11->device*[basecm11*]->roles*[scaleserver*]->advancedsettings*]% show
Parameter                      Value
------------------------------ ------------------------------------------------
Debug2                         no
Max Threads                    16
Power Operation Timeout        30
Connection Retry Interval      5
Log File                       /var/log/cm-scale.log
Pin Queues                     no
Mix Locations                  yes
Failed Node Is Healthy         no
```

```
Azure Disk Image Name            images
Azure Disk Container Name        vhds
Azure Disk Account Prefix
Node Selection                   Alphabetically
Node Selection Uptime Period     2w
```

The advanced settings are:

- `Debug2`: Enable printing very low level debug messages in the log file. This setting must be used with caution because it leads to a rapid increase in the log file size.
- `Max Threads`: Maximum number of threads for sequential RPCs to CMDaemon.
- `Power Operation Timeout`: Power operation RPC timeout, in seconds.
- `Connection Retry Interval`: Connection to CMDaemon retry interval, in seconds.
- `Log File`: Path to the log file (Default `/var/log/cm-scale.log`).
- `Pin Queues`: Pin workloads to their queue nodes.
- `Mix Locations`: Allow workload to be be offered to different locations (cloud and local).
- `Failed Node Is Healthy`: Do not start a new node instead of a failed one.
- `Azure Disk Image Name`: Image name for Azure disks.
- `Azure Disk Container Name`: Container name for Azure disks.
- `Azure Disk Account Prefix`: Prefix for randomly-generated Azure disk account names.
- `Node Selection`: Type of node selection used by Auto Scaler. The values this can take are:
  - `Alphabetically`: This means that Auto Scaler picks next node to start according the node name.
  - `Randomly`: This means that the nodes are picked randomly, which helps make the node usage more even.
  - `Uptime`: This means that the nodes that have been used the least amount of time are picked first.
- `Node Selection Uptime Period`: If `Node Selection` is set to `uptime`, then the `Node Selection Uptime Period` is the time period from now into the past, over which Auto Scaler calculates the total uptime for nodes. So, with the default setting of `2w` this means that the uptime of nodes is calculated over the last 2 weeks.

An overview of the parameters and submodes is given next. An example showing how they can be configured is given afterwards, in section 8.3.

ScaleServer Role Global Parameters

The `ScaleServer` role has the following global parameters for controlling the `cm-scale` service itself:

- `Debug`: Print debug messages to the log.
- `Dry Run`: If set, then the service runs in dry run mode. In this mode it may claim that actions have been carried out on the nodes that use the `cm-scale` service, however, no action is actually carried out on nodes. This mode is useful for demonstration or debug purposes
- `Run Interval`: interval, in seconds, between `cm-scale` decision-making

ScaleServer Role Submodes

Within the ScaleServer role are the following three submodes:

- advancedsettings: allows some advanced properties to be set for cm-scale, using the parameters displayed on page 481.
- resourceproviders: defines the nodes used by cm-scale. More explicitly, this submode is used to define resource provider objects. The resource providers can be added as static or dynamic types, and can then have nodes and settings defined within them. The nodes allocated to these resource provider objects are what provide resources to cm-scale when that resource provider is requested.
- engines: define the engines used by cm-scale. This can be an instance of the type hpc, generic, or kubernetes (page 489).
  - trackers (within engines submode): define the trackers used by cm-scale (page 490)

The parameters are enforced only when the next decision-making iteration takes place.

8.2.2 Resource Providers

The cm-scale service allows nodes to change state according to the workload demand. These managed nodes are defined by the administrator within the resourceproviders submode of ScaleServer. NVIDIA Base Command Manager 11 supports two types of resource providers: static and dynamic node providers.

Static Node Provider

When managed nodes are well-known and will not be extended or shrunk dynamically, then a static node provider can be used. Specifying settings for the static node provider allows cm-scale to power on, power off, or re-purpose nodes, based on nodegroups or a list of nodes nodes specified with a node list syntax (page 75).

The static node provider supports the following properties:

- Enabled: The static node provider is currently enabled.
- Nodes: A list of nodes managed by cm-scale. These can be regular local compute nodes (nodes) or cluster extension cloud compute nodes (cnodes). For the purposes of this section on cm-scale, these compute nodes can conveniently be called nodes and cnodes. Since compute nodes are typically the most common cluster nodes, significant resources can typically be saved by having the cm-scale service decide on whether to bring them up or down according to demand.
  - cnodes can be cloned and terminated as needed. Cloning and terminating saves on cloud storage costs associated with keeping virtual machine images.
  - regular local compute nodes can be started and stopped as needed. This reduces power consumption.
- Nodegroups: List of node groups (section 2.1.4) with nodes to be managed by cm-scale. Node groups are classed into *types*. The class of *node group types* is independent of the class of *node types*, and should not be confused with it.

  Node types are shown in the first column of the output of the default list command in device mode (page 55). The node types that can be managed by cm-scale are physicalnode and cloudnode.
- Priority: The provider priority. Nodes in the pool of a provider with a higher priority are used first by workloads. By default a resource provider has a priority value 0. These priority values should not be confused with the fairsharing priorities of page 488.

Dynamic Node Provider

When managed nodes can be cloned or removed from the configuration, then a `dynamic` node provider should be used. A compute node that is managed by `cm-scale` as a `dynamic` node provider is configured as a template node within the `dynamic` submode of the `ScaleServer` role.

The `dynamic` node provider supports the following properties:

- `Template Node`: A node that will be used as a template for cloning other nodes in the pool. The following restrictions apply to the template node:
  - A workload manager client role must be assigned with a positive number of slots.
  - New node names should not conflict with the node names of nodes in a `nodegroup` defined for the queue.
  - A specific template node is restricted to a specific queue.

  A template node only has to exist as an object in BCM, with an associated node image. A template node does not need to be up and running physically in order for it to be used to create clones. Sometimes, however, an administrator may want it to run too, like the other nodes that are based upon it, in which case the `Start Template Node` and `Stop Template Node` values apply.
  - `Start Template Node`: The template node specified in the `Template Node` parameter is also started automatically on demand.
  - `Stop Template Node`: The template node specified in the `Template Node` parameter is also stopped automatically on demand.

  An alternative to a template node is to use a snapshot (Chapter 3.4 of the *Cloudbursting Manual*), for greater cloud node startup speed.
- `Never Terminate`: Number of cloud nodes that are never terminated even if no jobs need them. If there are this number or fewer cloud nodes, then `cm-scale` no longer terminates them. Cloud nodes that cannot be terminated can, however, still be powered off, allowing them to remain configured in BCM. As an aside, local nodes that are under `cm-scale` control are powered off automatically when no jobs need them, regardless of the `Never Terminate` value.
- `Never Terminate Nodes`: A list of nodes specified with a node list syntax (page 75). These cloud nodes are never terminated, even if no jobs need them. Cloud nodes that cannot be terminated can, however, still be powered off. The nodes must already exist in the BCM configuration when `Never Terminate Nodes` is configured.
- `Enabled`: Node provider is currently enabled.
- `Priority`: Node provider priority.
- `Node Range`: Range of nodes that can be created and managed by `cm-scale`.
- `Network Interface`: Which node network interface is changed on cloning (incremented).
- `Remove Nodes`: Should the new node be removed from BCM when the node terminates? If the node is not going to be terminated, but just stopped, then it is never removed.
- `Leave Failed Nodes`: If nodes are discovered to be in a state of `INSTALLER_FAILED` or `INSTALLER_UNREACHABLE` (section 5.5.4) then this setting decides if they can be left alone, so that the administrator can decide what do with them later on.
- `Default Resources`: List of default resources, in format [*name=value*].
  - `cpu`: value is the number of CPUs
  - `mem`: value is in bytes

  These must be set when no real node instance is associated with a node defined in BCM.

Extra Nodes Settings For Node Providers

Both the `dynamic` and `static` node providers support extra node settings. If configured, then `cm-scale` can start the extra nodes before the first workload is started, and can stop them after the last job from the managed queue is finished.

The most common use case scenario for extra nodes in the case of cloud nodes is a cloud director node. The cloud director node provisions cloud compute nodes and performs other management operations in a cloud.

In the case of non-cloud non-head nodes, extra nodes can be, for example, a license server, a provisioning node, or an additional storage node.

The configuration settings include:

- `Extra Nodes`: A list of extra nodes.
- `Extra Node Idle Time`: The maximum time, in seconds, that extra nodes can remain unused. The `cm-scale` service checks for the existence of queued and active workloads using the extra node, when the time elapsed since the last check reaches `Extra Node Idle Time`. If there are workloads using the extra node, then the time elapsed is reset to zero and a time stamp is written into the file `cm-scale.state` under the directory set by the `Spool` role parameter. The time stamp is used to decide when the next check is to take place. Setting `Extra Node Idle Time=0` means the extra node is stopped whenever it is found to be idle, and started again whenever workloads require it, which may result in a lot of stops and starts.
- `Extra Node Start`: Extra node is started by `cm-scale` before the first compute node is started.
- `Extra Node Stop`: Extra node is stopped by `cm-scale` after the last compute node stops.

Additional Settings For Node Providers

All of the node providers include the following settings that allows the Auto Scaler behavior to be tuned:

- `Keep Running`: Nodes that should not be stopped or terminated even if they are unused (range format).
- `Shutdown Before Power Off`: Shutdown nodes instead of just power off, and wait until a set timeout before doing a hard power off.
- `Shutdown Timeout`: Shutdown timeout before powering off.
- `Allocation Prolog`: Script that is executed when a node is allocated to a workload.
- `Allocation Epilog`: Script that is executed when a node is deallocated.
- `Long starting node action`: Action that is applied to a *long starting node*. A long starting node is a node that takes too long to start. Options:
  - `none` (default)
  - `power off`
  - `terminate` (applied to dynamic node provider only)
- `Long starting node timeout`: How long Auto Scaler should wait before the action is applied for a long starting node.

The following table summarizes the default attributes in `cmsh` for the resource providers, along the `cmsh` path `cmsh->device[]->roles->scaleserver->resourceproviders[dynamic/static]`:

```
Parameter                    static                    dynamic
---------------------------  ------------------------  ----------------------
Name                         static                    dynamic
Revision
Type                         static                    dynamic
Enabled                      yes                       yes
Priority                     0                         0
Whole Time                   0                         0
Stopping Allowance Period    0                         0
Keep Running
Extra Node
Extra Node Idle Time         1h                        1h
Extra Node Start             yes                       yes
Extra Node Stop              yes                       yes
Allocation Prolog
Allocation Epilog
Allocation Scripts Timeout   10s                       10s
Nodes                                                  N/A
Template Node                N/A
Node Range                   N/A
Network Interface            N/A                       tun0
Start Template Node          N/A                       no
Stop Template Node           N/A                       no
Remove Nodes                 N/A                       no
Leave Failed Nodes           N/A                       yes
Never Terminate              N/A                       32
Never Terminate Nodes        N/A
Nodegroups                                             N/A
Default Resources            cpus=1                    cpus=1
Shutdown Before Power Off    yes                       yes
Shutdown Timeout             3m                        3m
Long starting node action    None                      None
Long starting node timeout   10m                       10m
```

In the preceding table, the entry N/A means that the parameter is not available for the corresponding resource provider.

8.2.3 Time Quanta Optimization

Time quanta optimization is an additional feature that `cm-scale` can use for further cost-saving with certain cloud providers.

For instance, a cloud provider may charge per whole unit of time, or *time quantum*, used per cloud node, even if only a fraction of that unit of time was actually used. The aim of BCM's time quanta optimization is to keep a node up as long as possible within the already-paid-for time quantum, but without incurring further cloud provider charges for a node that is not currently useful. That is, the aim is to:

- keep a node up if it is running jobs in the cloud
- keep a node up if it is not running jobs in the cloud, if its cloud time has already been paid for, until that cloud time is about to run out
- take a node down if it is not running jobs in the cloud, if its cloud time is about to run out, in order to avoid being charged another unit of cloud time

Time quanta optimization is implemented with some guidance from the administrator for its associated parameters. The following parameters are common for both `static` and `dynamic` node resource providers:

- `Whole time`. A compute node running time (in minutes) before it is stopped if no workload requires it. For example, the cloud provider may have a time quantum of 60 minutes. By default, BCM uses a value of `Whole Time`=0, which is a special value that means `Whole Time` is ignored. Ignoring it means that BCM does no time quanta optimization to try to optimize how costs are minimized, but instead simply takes down nodes when they are no longer running jobs.

- `Stopping Allowance Period`. A time (in minutes) just before the end of the `Whole Time` period, prior to which all power off (or terminate) operations must be started. The parameter associated with time quanta optimization is the `Stopping Allowance Period`. This parameter can also be set by the administrator. The `Stopping Allowance Period` can be understood by considering the *last call time period*. The last call time period is the period between the last call time, and the time that the next whole-time period starts. If the node is to be stopped before the next whole-time charge is applied, then the last call time period must be at least more than the maximum time period that the node takes to stop. The node stopping period in a cluster involves cleanly stopping many processes, rather than just terminating the node instance, and can therefore take some minutes. The maximum period in minutes allowed for stopping the node can be set by the administrator in the parameter `Stopping Allowance Period`. By default, `Stopping Allowance Period`=0. Thus, for nodes that are idling and have no jobs scheduled for them, only if the last call time period is more than `Stopping Allowance Period`, does `cm-scale` stop the node.

The preceding parameters are explained next.
Figure 8.20 illustrates a time line with the parameters used in time quanta optimization.

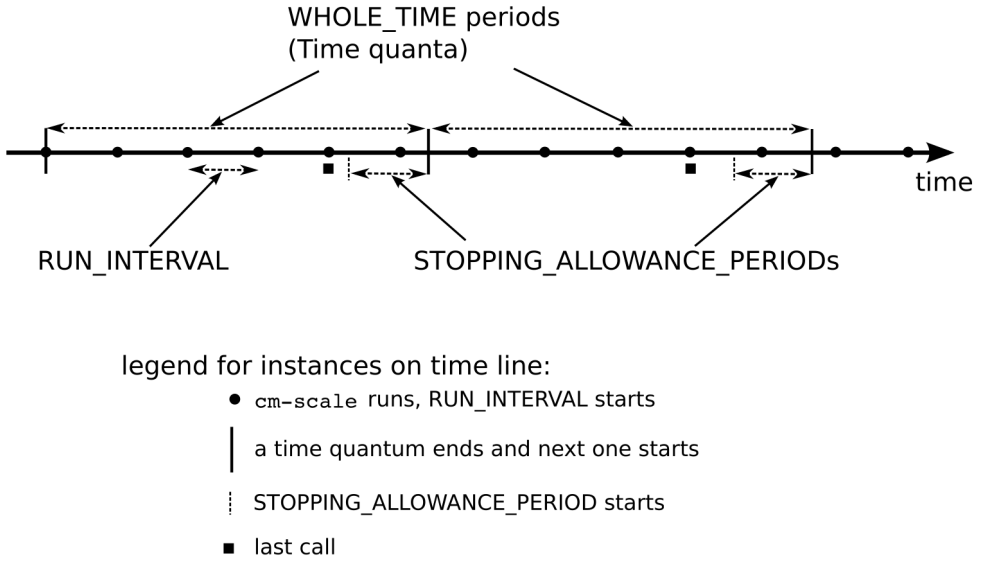


Figure 8.20: Time Quanta Optimization

The algorithm that `cm-scale` follows, with and without time quanta optimization, can now be described using the two parameters explained so far:

1. `cm-scale` as part of its normal working, checks every `Run Interval` seconds to see if it should start up nodes on demand or shut down idling nodes.

2. If it sees idling nodes, then:

   (a) If `Whole Time` has not been set, or is 0, then there is no time quanta optimization that takes place. The `cm-scale` service then just goes ahead as part of its normal working, and shuts down nodes that have nothing running on them or nothing about to run on them.

(b) If a non-zero `Whole Time` has been set, then a time quanta optimization attempt is made. The `cm-scale` service calculates the time period until the next time quantum from public cloud starts. This time period is the current *closing time period*. Its value changes each time that `cm-scale` is run. If

- the current closing time period is long enough to let the node stop cleanly before the next time quantum starts, and
- the next closing time period—as calculated by the next `cm-scale` run but also running within the current time quantum—is not long enough for the node to stop cleanly before the next quantum starts

then the current closing time period starts at a time called the *last call*.

In drinking bars, the last call time by a bartender allows some time for a drinker to place the final orders. This allows a drinker to finish drinking in a civilized manner. The drinker is meant to stop drinking before closing time. If the drinker is still drinking beyond that time, then a vigilant law enforcement officer will fine the bartender.

Similarly, the last call time in a scaling cluster allows some time for a node to place its orders to stop running. It allows the node to finish running cleanly. The node is meant to stop running before the next time quantum starts. If the node is still running beyond that time, then a vigilant cloud provider will charge for the next whole time period.

The last call time is the last time that `cm-scale` can run during the current whole-time period and still have the node stop cleanly within that current whole-time period, and before the next whole-time period starts. Thus, when `Whole Time` has been set to a non-zero time:

i. If the node is at the last call time, then the node begins with stopping

ii. If the node is not at the last call time, then the node does not begin with stopping

The algorithm goes back again to step 1.

8.2.4 Fairsharing Priority Calculation And Node Management

At intervals of `Run Interval`, `cm-scale` collects workloads using trackers configured in the `ScaleServer` role, and puts all the workloads in a single internal queue. This queue is then sorted by priorities. The priorities are calculated for each workload using the following fairsharing formula:

$$p_{ij} = k_1 \times a_i + k_2 \times b_j + k_3 \times c_j \tag{8.1}$$

where:

p_{ij} is the global priority for the i-th workload of the j-th engine. Its value is used to re-order the queue.

k_1 is the age factor. This is the `agefactor` parameter that can be set via `cmsh` in the `engine` submode of the `ScaleServer` role. Usually it has the value `1`.

a_i is the age of the workload. That is, how long has passed since the i-th job submission, in seconds. This typically dominates the priority calculation, and makes older workloads a higher priority.

k_2 is the external priority factor. It is a floating point number in the range $[0, 1]$, and is the`External Priority Factor` parameter that can be set via `cmsh` in the `engine` submode of the `ScaleServer` role.

b_i is the workload priority retrieved from the engine.

k_3 is the engine factor. It is a floating point number in the range $[0, 1]$, and is the `enginefactor` parameter in the `engine` submode of the `ScaleServer` role.

c_j is the engine priority. This is the `priority` parameter in the `engine` submode of the `ScaleServer` role.

When all the workload priorities are calculated and the queue is re-ordered, then `cm-scale` starts to find appropriate nodes for workloads. The workloads are selected in order, from the top of the queue

where the higher priority workloads are, to the bottom. This way a higher priority engine has a greater chance of getting nodes for its workloads than a lower priority engine.

The factors k_1, k_2 and k_3 in the equation 8.1 allow the significance of the related priority value in the final result to be controlled. For example if only the priority fetched from the engine should be taken into account, then k_1 and k_3 should be set to `0`, and k_2 to `1`. Or, for example, when both the age and engine priorities should be treated as equally important, then k_1 and k_3 can be set to `0.5` and k_2 to `0`.

8.2.5 Engines

Each workload engine considered by `cm-scale` must be configured within the `engines` submode within the `ScaleServer` role. NVIDIA Base Command Manager 11 supports the following workload engines:

- Slurm
- LSF

Engines can be of three types:

- `hpc`: for all HPC (High Performance Computing) workload managers
- `kubernetes`: for Kubernetes
- `generic`: for a generic type

Common Parameters For The `cm-scale` Engines

All three engine types have the following parameters and submode in common, although their values may differ:

- `Workloads Per Node`: The number of workloads, Kubernetes jobs, or individual Kubernetes pods without a controller, that can be scheduled to run on the same node at the same time.
  - For a Kubernetes engine this parameter restricts the number of jobs or individual pods per node. It does not restrict the total number of pods that can be run per node by the `cm-scale` scheduler. The parameter is taken into consideration by the `cm-scale` scheduler when it is searching for new nodes to start up, and does not configure Kubernetes itself.

    For example, a Kubernetes job, or Job with a capital 'J' in Kubernetes terminology, may consist of many pods. Then, if `Workload Per Node` is, for example, 2, then only 2 Jobs are run on the node.

    The number of pods is also taken into account by `cm-scale`, but this number is taken from the `kubelet` role, where the `Max Pods` option can be set. If the role is not assigned to a node, using a configuration overlay, cateogry, or node, then `cm-scale` assumes that there is no possibility for any pods to run on the node.
- `Priority`: The engine priority
- `Age Factor`: Fairsharing coefficient for workload age
- `Engine Factor`: Fairsharing coefficient for engine priority
- `External Priority Factor`: Fairsharing coefficient for external priority significance
- `Trackers`: Enters the workload trackers submode

Non-common parameters for the `cm-scale` engines:

- For the `hpc` engine:
  - `WLM Cluster`: A workload manager cluster name. The name is set during workload manager setup as the instance name. In `cmsh` the WLM cluster names are listed under `wlm` mode. In Base View they can be seen along the navigation path `HPC > Workload Management Clusters`.
- For the `kubernetes` engine, the following parameters can be set:
  - `Cluster`: These are the Kubernetes clusters for which pods are to be tracked. BCM allows multiple Kubernetes clusters to run on a single compute cluster. Kubernetes must be already set up before this setting is configured.
  - `CPU Busy Threshold`: The CPU load is a value that can range from 0 to 1. The `CPU Busy Threshold` value defines if the node is too busy for new pods. Its default value is: `0.9`.
  - `Memory Busy Threshold`: The Memory load is a value that can range from 0 to 1. The `Memory Busy Threshold` defines if the node is too busy for new pods. Its default value is: `0.9`.

The CPU and Memory thresholds configured in the Kubernetes engine help `cm-scale` to decide when more nodes are needed. But `cm-scale` also retrieves the number of pods that are already running on the node and compares it with the `Max Pods` parameter that is configured in the `kubelet` role assigned to the node via at the configuration overlay level, category level, or node level. If the number of running pods is already equal or greater than the value of `Max Pods`, then (from the `cm-scale` point of view) the node cannot fit more pods, which means that a new node is needed.

8.2.6 Trackers

A workload tracker is a way to specify the workload and its node requirements to `cm-scale`. For HPC, the tracker may be associated with a specific queue, and `cm-scale` then tracks the jobs in that queue.

One or more trackers can be named and enabled within the `trackers` submode, which is located within the `engines` submode of the `ScaleServer` role. A queue (for workload managers) can be assigned to each tracker.

Example

There are three types of tracker objects supported in NVIDIA Base Command Manager 11:

- `queue`: Used with an HPC type engine, where each workload (job) is associated with a particular queue. The attribute `Type` takes the value `ScaleHpcQueueTracker`, and the attribute `Queue` is set to the queue name for the job.
- `namespace`: Used with a `kubernetes` type engine.
- `generic`: Used with a `generic` type engine.

The following settings are common for both types of trackers:

- `Enabled`: Enabled means that workloads from this tracker are considered by `cm-scale`.
- `Allowed Resource Providers`: Only the specified resource providers (in the scaleserver role) will be used for a workload of this tracker (if empty than all allowed).
- `Assign Category`: A node category name that should be assigned to the managed nodes. When a node is supposed to be used by a workload, then `cm-scale` should assign the node category to that node. If the node is already running, and has no workloads running on it, but its category differs from the category specified for the jobs of the queue, then the node is drained, stopped and restarted on the next decision-making iteration of `cm-scale`, and takes on the assigned category. Further details on this are given in the section on dynamic nodes re-purposing, page 506.

- `Primary Overlays`: A list of configuration overlays.

  If a workload is associated with the tracker for which the overlays are specified by `Primary Overlays`, then BCM-managed nodes are appended to those configuration overlays by `cm-scale`. This takes place after the node is removed from the previous overlays that it is associated with.

  If the node is already running, but has not yet been appended to the overlays specified for the workloads of the tracker, then the node is restarted when no other workloads run on the node, before going on to run the workloads with the new overlays.

  - When a workload is associated with the tracker that has `Primary Overlays` set, then the pool of `cm-scale`-managed nodes is checked.

    The check is to decide on if a node is to be made available for the workload.

    If the node is appended to the `Primary Overlays` already and is not running workloads, then `cm-scale` simply hands over a workload from the tracker to run on the node.

    If the node is not appended to the `Primary Overlays` already, and is not running workloads, then `cm-scale` prepares and boots the node as follows:

    * the node is drained and rebooted if it is up, or
    * the node is undrained and merely booted if it is not up

    The node is removed from any previous overlays that it was with, before booting up, and it is appended to the new overlays of `Primary Overlays`.

- The threshold settings:

  - `Queue Length Threshold`: number of pending workloads that triggers cloudbursting.
  - `Age Threshold`: workload pending time threshold, in seconds, that triggers cloudbursting for this workload.

  The queue length and age thresholds allow the administrator to set when `cm-scale` starts or creates cloudbursting nodes. Both thresholds can be used at the same time, or just one of them can be used and the other can be ignored by setting it to `0`.

  If the queue length threshold is set, then `cm-scale` ignores pending workloads that are located higher (added later) than the threshold in the managed queue.

  Example

  Assuming there are 5 jobs in the queue, with job IDs 1, 2, 3, 4, and 5, where the 1st one is the first in the queue. If the queue length threshold is 3, then only jobs 1, 2 and 3 are taken into account, while jobs 4 and 5 are ignored.

  Example

  If the age threshold is set to `100`, then only workloads older than 100 seconds are taken into account, while younger jobs are ignored.

The `queue` type tracker has only one parameter specific to the tracker: `Queue`. This is set to the workload queue that is being tracked.

Namespace Tracker

The namespace tracker of `cm-scale` is used to track Kubernetes workloads. It tracks Kubernetes jobs (via its Job controllers) and tracks individual pods. It does not start new nodes for pending pods owned by other types of Kubernetes pod controllers, such as ReplicaSet, DaemonSet, and so on. If non-Job controllers are running, then `cm-scale` will not stop or terminate those nodes.

The tracker settings in `cmsh` or Base View include additional parameters that are in common with other trackers:

1. `Controller Namespace`: Tracks the Kubernetes namespace name. Only Kubernetes workloads from this namespace are tracked. To track more than one namespace, one tracker must be created per namespace.
2. `Object`: Type of Kubernetes objects to track. BCM supports the following object types:
   (a) `Job`: A Kubernetes Job controller type represents one or several pods that are expected to eventually terminate. The controller nature makes this type of Kubernetes workload very suited to dynamic data centers.
   (b) `Pod`: Individual pod, without any controller.

If the specified namespace does not exist in Kubernetes, then the tracked jobs or individual pods in this namespace are ignored by `cm-scale`.

Generic Engine And Tracker

The `cm-scale` service is able to deal with workloads that use various workload types. In order to add suuport of a new type of workload, the administrator

- adds an engine of type `generic`
- adds one or more trackers of type `generic` to the ScaleServer role
- implements Tracker and Workload classes in the Python programming language

When `cm-scale` starts a new iteration, it re-reads the engines and trackers settings from the `ScaleServer` role, and searches for the appropriate modules in its directories. In the case of a custom tracker, the module is always loaded according to the tracker handler path. When the tracker module is loaded, `cm-scale` requests a list of workloads from each of the tracker modules. So, the aim of the tracker module is to collect and provide the workloads to `cm-scale` in the correct format.

The path to the tracker module should be specified in the handler parameter of the generic tracker entity using `cmsh` or Base View as follows:

Example

```
[root@basecm11 ~]# cmsh
[basecm11]% device roles master
[basecm11->device[basecm11]->roles]% use scaleserver
[basecm11->device[basecm11]->roles[scaleserver]]% engines
...->roles[scaleserver]->engines]% add generic myengine
...*]->roles*[scaleserver*]->engines*[myengine*]]% trackers
...*[myengine*]->trackers]% add generic mytracker
...*[mytracker*]]% set handler /cm/local/apps/cm-scale/examples/custom_tracker/tracker.py
...*]->roles*[scaleserver*]->engines*[myengine*]->trackers*[mytracker*]]% commit
...->roles[scaleserver]->engines[myengine]->trackers[mytracker]]%
```

In the preceding example, the handler file `.../examples/tracker.py` is an example that is provided with the `cm-scale` package. Another example module file provided with the package is `.../examples/workload.py`, which implements the `ExampleWorkload` class. Together, the two examples can be used

to generate any amount of simple workloads during each `cm-scale` iteration. It is recommended to use the tracker and workload classes as templates for custom tracker and workload modules created by the administrator.

The generic engine does not have any specific parameters associated with its type. It only has parameters common to all of the engine types.

If the generic engine is configured with at least one generic tracker in its role, then `cm-scale` loads the handler module and uses two functions that are implemented in the class. The class name can be chosen arbitrarily, but should contain the string "`Tracker`", without the quotes. The two class functions used are:

1. `__init__`: initializes the tracker object. This can be omitted if there are no additional data values to initialize.
2. `get_workloads`: returns a list of objects belonging to a new class inherited from the `Workload` class. This new class should be created by the administrator.

The new workload class must provide the following functions and properties. The class name can be chosen arbitrarily:

1. `__init__`: initializes the workload object.
2. `to_string`: returns a string that identifies the workload. This is printed to the log file.
3. `begin_timestamp`: property that returns a unix timestamp that should be >0 if the workload is not allowed to start before that time. If it is 0 then it is ignored by `cm-scale`.

For example, the following very simple tracker and workload classes can be implemented:

Example

```
class ExampleTracker(Tracker):
    def get_workloads(self):
        return [ExampleWorkload(self)]

class ExampleWorkload(Workload):
    def __init__(self):
        Workload.__init__(self, tracker)
        self.set_id("1")
        self._update_state()
        self._update_age()
        self._update_resources()

    def to_string(self):
        return "workload %s" % self._id

    def _update_state():
        self._set_pending()

    def _update_age(self):
        self._age = 0

    def _update_resources(self):
        node_res = NodeResource("*")

        cpus_res = CpusResource(1)
        node_res.add_resource(cpu_res)
```

```
        engine_res = EngineResource("myengine")
        node_res.add_resource(engine_res)

        self.add_resource(node_res)
```

The classes should be located in different files, as Python module files. It is recommended, but not required, to keep both the files in the same directory. The `ExampleWorkload` class initializes the workload object with a state, age, and required resources. These values are described next.

State: The state can be `pending`, `running` or `failed`, which can be set with these appropriate functions:

- `self._set_pending()`
- `self._set_running()`
- `self._set_failed()`

If the state is `running`, then the workload is treated as one that occupies the nodes defined in the resources list. Each NodeResource object in the resources thus represents one occupied node.

If the state is `pending`, then the workload is treated as one that waits for free nodes. In this case `cm-scale` tries to find (start or clone) some more nodes in order to allow the workload engine to start this workload.

The `failed` workload state is considered by `cm-scale` as exceptional. Such a workload is logged in the log file, but is not considered when `cm-scale` decides what nodes to start or clone. Any other workload state is also ignored by `cm-scale`.

Age: The `age` defines how many seconds the workload is waiting for its resources since being added to the engine. Usually the engine can provide such information, so in the example `age` is set to `0`, which means the workload has been added to the workload engine just now. The `age` is used in the fairsharing workload priority calculation (page 488). The value of `age` is not re-calculated by `cm-scale` after a while. This means that the number that the module sets in the class is used during the iteration, and then forgotten by `cm-scale` until the workload object is recreated from scratch on the next iteration.

Resources: The Workload class (a base class) includes the `resources` property with list types. This list includes resource objects that are used by `cm-scale` in order to find appropriate nodes for the workload. The top level resource type is always NodeResource. There can be one or several node resources requested by the workload.

If the node names are known, then one NodeResource object is created per compute node.

Otherwise a single NodeResource object is used as many times as the number of requested nodes, with the name set to `*`, which is treated by `cm-scale` as any suitable node. The number of nodes can be set in the NodeResource object with the `set_amount(number)` function of the resource.

In the preceding example one (any) node resource is added to the workload request, and the requirement for CPU (cores) number is set to 1. The engine resource is used in order to restrict the running of workloads from different engines to one node. Thus if a node has this resource assigned, then the node can take on the workload. If no engine resource is assigned to the node, then it can also take on the workload, but the engine resource of the workload is assigned to the node before other workloads are considered.

The resource types that can be added to the workload are defined in the Python module `core/resource.py`:

- `NodeResource`: top level resource, contains all other resources.
- `CpusResource`: defines the number of cpu cores required or already used by the workload.

- CategoryResource: node category required by the workload.
- OverlayResource: required configuration overlay.
- QueueResource: HPC queue that the workload (job) belongs to. Used only with engines that support queues.
- EngineResource: engine name that the workload belongs to.
- FeatureResource: required node feature (node property, in other terminology) that should be supported by the engine.

Custom resource types are not supported for now.

In order to drain a node in the custom engine before the node is stopped, and to undrain it before the node is started, the administrator can write and configure three scripts:

- Drain script: called before a node is drained by Auto Scaler.
- Undrain script: called before node is undrained by Auto Scaler.
- Drain status script: called when Auto Scaler retrieves information about the current node drain status.

Either all of the three scripts must be configured, or none of them.

It is useful to drain and undrain the nodes in order to ensure that the engine does not start new jobs in time period between the instant that Auto Scaler decides to stop the node, and the instant that the actual power operation is performed.

The scripts are configured in the file:

```
/cm/local/apps/cm-scale/lib/python3.12/site-packages/cmscale/config.py
```

with the GENERIC_DRAIN_COMMANDS parameter appended to the opts dictionary:

Example

```
"GENERIC_DRAIN_COMMANDS": {
  "MyEngine":
    {"drain": "/cm/local/apps/cm-scale//examples/custom_drain/drain.py",
     "undrain": "/cm/local/apps/cm-scale/examples/custom_drain/undrain.py",
     "status": "/cm/local/apps/cm-scale/examples/custom_drain/drainstatus.py"}
},
```

Here, for each generic engine, a new dictionary is created that includes three items that correspond to, and specify, the script paths. In the preceding example MyEngine is the engine name, and should be the same as that defined in the ScaleServer role. If more then one generic engine is used then all of them can be added to GENERIC_DRAIN_COMMANDS.

It should be noted that if GENERIC_DRAIN_COMMANDS is defined in config.py, then CMDaemon does not drain, or undrain, via cm-scale.

All three scripts accept the same set of parameters, following the form:

<script name> *<engine name>* *<host name>* [*host name ...*]

Example

```
drain.py MyEngine node001 node002 node003
```

Each of those three scripts print the following information to standard output:

- stdout: JSON structure that represents a map: hostname -> latest (new) drain status. For example:

Example

```
{"node001": 2, "node002": 2, "node003": 2}
```

Here the numbers are enum values defined in pythoncm in the DrainResult class.

- `stderr`: debug logs that are appended to `cm-scale.log`.

Enabling Node Shutdown

By default, `cm-scale` powers off nodes belonging to a resource pool once there is no more workload for them. Resource providers can also enable shutdown, to allow the node to terminate gracefully. Shutdown has two options that set its behavior directly:

- `Shutdown Enable`: If set to `yes`, then the shutdown command is run to terminate the system services first, and after that a command is run to power off the system. A waiting time of `Shutdown Timeout` seconds takes place between the two commands.
- `Shutdown Timeout`: The number of seconds to wait before powering off a node that is in a shutdown state.

It may take more than `Shutdown Timeout` seconds for a node to power off, depending on the `Run Interval` setting. For example, if `Shutdown Timeout` is 60, and `Run Interval` 50, then effectively the `Shutdown Timeout` is 100, because the power off event only happens during an iteration execution of `cm-scale`.

Multi-partition Slurm jobs

Slurm allows a user to submit a job that requests multiple queues. Auto Scaler detects such jobs and tries to start nodes for the job. The queue with the maximum priority is first considered. If no nodes are found in that partition, then the next requested partition in order of priority, is considered.

The queue priority is taken from Slurm partition `PriorityTier` parameter, accessible via `cmsh` or Base View.

- In `cmsh`, the queue priorities can be seen from within `jobqueue` mode. In the following example there are 3 queues with different priorities:

Example

```
[root@basecm11 ~]# cmsh
[basecm11]% wlm jobqueue; list
Name (key)   Nodes
------------ ------------------------
defq         node001..node005
medq         node006..node009
topq         node010..node012
[basecm11->wlm[slurm]->jobqueue]% get defq prioritytier; get medq prioritytier; get topq prioritytier
1
5
10
[basecm11->wlm[slurm]->jobqueue]% use defq; help set | grep tier
   prioritytier ........ Jobs submitted to a partition with a higher priority tier value will be
                         dispatched before pending jobs in partition with lower priority tier value
```

- In Base View the navigation path for the queue `defq` is:
  `HPC > Workload Management Clusters > slurm` > `Job Queues > defq` > `Priority Tier`

If multiple queue trackers are configured, and if queues are requested by the Slurm job that are tracked by different trackers, then only one tracker sees the job—the tracker for which tracking queue priority is the highest.

If two queues have the same priority, then the next selection criterion is the order of placement of its trackers (section 8.2.6) in the trackers list.

For example, if the Auto Scaler (as defined by the scaleserver role) is running on the head node basecm11, and if the trackers are, for example, mytracker and secondtracker, and if the engine is, for example, myengine, then the order of placement can be listed in cmsh via the path indicated by:

Example

```
[basecm11->device[basecm11]->roles[scaleserver]->engines[myengine]->trackers]% list
Name (key)       Enabled
---------------- -------
mytracker        yes
secondtracker    yes
```

8.3 Examples Of cm-scale Use

8.3.1 Simple Static Node Provider Usage Example

The example session that follows explains how a static node provider (page 483) can be configured and used with cm-scale. The session considers a default cluster with a head node and 5 regular nodes which have been previously defined in the BCM configuration. 3 of the regular nodes are powered down at the start of the run. The power control for the nodes must be functioning properly, or otherwise cm-scale cannot power nodes on and off.

The head node has the Slurm server role by default, and the regular nodes run with the Slurm client role by default. So, on a freshly-installed cluster, the roleoverview command should show something like:

Example

```
[basecm11->device[basecm11]]% roleoverview | head -2; roleoverview | grep slurm
Role             Nodes                      Categories Configuration Overlays             Nodes up
---------------- -------------------------- ---------- ---------------------------------- --------
slurmaccounting  basecm11                              slurm-accounting                   1 of 1
slurmclient      node001..node005           default    slurm-client                       2 of 5
slurmserver      basecm11                              slurm-server                       1 of 1
slurmsubmit      basecm11,node001..node005 default    slurm-submit, wlm-headnode-submit 3 of 6
```

A test user, fred can be created by the administrator (section 6.2), and an MPI hello executable based on the hello.c code (from section 3.5.1 of the *User Manual*) can be built:

Example

```
[fred@basecm11 ~]$ module add shared openmpi/gcc/64 slurm
[fred@basecm11 ~]$ mpicc hello.c -o hello
```

A batch file slurmhello.sh (from section 5.3.1 of the *User Manual*) can be set up. Restricting it to 1 process per node so that it spreads over nodes easier for the purposes of the test can be done with the settings:

Example

```
[fred@basecm11 ~]$ cat slurmhello.sh
#!/bin/sh
#SBATCH -o my.stdout
```

```
#SBATCH --time=30 #time limit to batch job
#SBATCH --ntasks=1
#SBATCH --ntasks-per-node=1
module add shared openmpi/gcc/64/ slurm

mpirun /home/fred/hello
```

The user fred can now flood the default queue, defq, with the batch file:

Example

```
[fred@basecm11 ~]$ while (true); do sbatch slurmhello.sh; done
```

After putting enough jobs into the queue (a few thousand should be enough, and keeping it less than 5000 would be sensible) the flooding can be stopped with a ctrl-c.

The activity in the queue can be watched:

Example

```
[root@basecm11 ~]# watch "squeue | head -3 ; squeue | tail -3"

Every 2.0s: squeue | head -3 ; squeue | tail -3              Thu Sep 15 10:33:17 2016

             JOBID PARTITION     NAME     USER ST       TIME  NODES NODELIST(REASON)
              6423      defq slurmhel     fred CF       0:00      1 node001
              6424      defq slurmhel     fred CF       0:00      1 node002
              6572      defq slurmhel     fred PD       0:00      1 (Priority)
              6422      defq slurmhel     fred  R       0:00      1 node001
              6423      defq slurmhel     fred  R       0:00      1 node002
```

The preceding indicates that node001 and node002 are being kept busy running the batch jobs, while the remaining nodes are not in use. The ST column is a status column, and indicates whether the job is CF (configuring), PD (pending), or R (running).

Abusing squeue in a loop like this is regarded as a bad practice, and doing it should be minimized.

The administrator can check on the job status via the job metrics of cmsh too, using the options to the filter command, such as --pending or --running:

Example

```
[root@basecm11 ~]# cmsh -c "wlm use slurm; jobs; watch filter --running -u fred"
Every 2.0s: filter --running -u fred                          Wed May  7 12:50:05 2017
Job ID Job name      User Queue Submit time Start time End time Nodes           Exit code
------ ------------- ---- ----- ----------- ---------- -------- --------------- ---------
406    slurmhello.sh fred defq  16:16:56    16:27:21   N/A      node001,node002  0
```

and eventually, when jobs are no longer running, it should show something like:

```
[root@basecm11 ~]# cmsh -c "wlm use slurm; jobs; watch filter --running -u fred"
Every 2.0s: filter --running                                   Wed May  7 12:56:53 2017
No jobs found
```

So far, the cluster is queuing or running jobs without cm-scale being used.

The next steps are to modify the behavior by bringing in cm-scale. The administrator assigns the ScaleServer role to the head node. Within the role a new static node provider, Slurm engine, and queue tracker for the defq are set as follows:

Example

```
[root@basecm11 ~]# cmsh
[basecm11]% device roles master
[basecm11->device[basecm11]->roles]% use scaleserver
[basecm11->device[basecm11]->roles[scaleserver]]% resourceproviders
...->roles[scaleserver]->resourceproviders]% add static pool1
...*]->roles*[scaleserver*]->resourceproviders*[pool1*]]% set nodes node001..node005
...*]->roles*[scaleserver*]->resourceproviders*[pool1*]]% commit
...]->roles[scaleserver]->resourceproviders[pool1]]% ..;..
...]->roles[scaleserver]]% engines
...]->roles[scaleserver]->engines]% add hpc slurm1
...*]->roles*[scaleserver*]->engines*[slurm1*]]% set wlmcluster slurm
...*]->roles*[scaleserver*]->engines*[slurm1*]]% trackers
...*]->roles*[scaleserver*]->engines*[slurm1*]->trackers]% add queue tr1
...*]->roles*[scaleserver*]->engines*[slurm1*]->trackers*[tr1*]]% set queue defq
...*]->roles*[scaleserver*]->engines*[slurm1*]->trackers*[tr1*]]% commit
...->roles[scaleserver]->engines[slurm1]->trackers[tr1]]%
```

The nodes node001..node005 should already be in the queue defq, as assigned to them by default when they were assigned the SlurmClient role. With these settings, they can now be powered up or down on demand by cm-scale service, depending on the number of jobs that are pending. When the new ScaleServer role is committed in cmsh or Base View, then the cm-scale service is started. If needed, the administrator can check the log file /var/log/cm-scale to see what the service is doing.

On each iteration cm-scale checks whether the node states should be changed. Thus after a while, the nodes node003..node005 are started. Once up, they can start to process the jobs in the queue too.

Watching the running jobs should show the newly-started nodes running too:

Example

```
[root@basecm11 ~]# cmsh -c "wlm use slurm; jobs ; watch filter --running"
Every 2.0s: filter --running                                Thu Apr 25 16:21:59 2024
Job ID Job name      User Queue Submit time Start time End time Nodes    Exit code
------ ------------- ---- ----- ----------- ---------- -------- -------- ----------
6147   slurmhello.sh fred defq  16:16:37    16:20:17   N/A      node004  0
6148   slurmhello.sh fred defq  16:16:37    16:20:17   N/A      node001  0
6149   slurmhello.sh fred defq  16:16:37    16:20:17   N/A      node003  0
```

Eventually, cm-scale finds that all jobs have been dealt with, and the nodes are then powered down.

High-availability And Using A Configuration Overlay For The ScaleServer Role

For high-availability clusters, where there are two head nodes, the scaleserver should run on the active head node. One labor-intensive way to set this up is to assign the service to both the head nodes, and match the scaleserver settings on both head nodes. A simpler way is to define a configuration overlay for the head nodes for the scaleserver. If the head nodes are basecm11-1 and basecm11-2, then a configuration overlay called basecm11heads can be created and assigned the service as follows:

Example

```
[basecm11-1]% configurationoverlay add basecm11heads
[basecm11-1->configurationoverlay*[basecm11heads*]]% append nodes basecm11-1 basecm11-2
[basecm11-1->configurationoverlay*[basecm11heads*]]% roles
[basecm11-1->configurationoverlay*[basecm11heads*]->roles]% assign scaleserver
[basecm11-1->configurationoverlay*[basecm11heads*]->roles*[scaleserver*]]%
```

The scaleserver can then be configured within the configuration overlay instead of on a single head as was done previously in the example of page 498. After carrying out a `commit`, the `scaleserver` settings modifications are then mirrored automatically between the two head nodes.

Outside the `scaleserver` settings, one extra modification is to set the `cm-scale` service to run on a head node if the head node is active. This can be done with:

Example

```
[basecm11-1->configurationoverlay[basecm11heads]->roles[scaleserver]]% device services basecm11-1
[basecm11-1->device[basecm11-1]->services]% use cm-scale
[basecm11-1->device[basecm11-1]->services[cm-scale]]% set runif active
[basecm11-1->device*[basecm11-1*]->services*[cm-scale*]]% commit
[basecm11-1->device[basecm11-1]->services]% device use basecm11-2
[basecm11-1->device[basecm11-2]->services]% use cm-scale
[basecm11-1->device[basecm11-2]->services[cm-scale]]% set runif active
[basecm11-1->device*[basecm11-2*]->services*[cm-scale*]]% commit
```

The result is a scaleserver that runs when the head node is active.

8.3.2 Simple Dynamic Node Provider Usage Example

The following example session explains how a dynamic node provider (page 484) can be configured and used with `cm-scale`. The session considers a default cluster with a head node and 2 regular nodes which have been previously defined in the BCM configuration, and also 1 cloud director node and 2 cloud compute nodes. The cloud nodes can be configured using `cm-cluster-extension`. Only the head node is running at the start of the session, while the regular nodes and cloud nodes are all powered down at the start of the run.

At the start, the device status shows something like:

Example

```
[basecm11->device]% ds
eu-west-1-cnode001 ....... [  DOWN  ] (Unassigned)
eu-west-1-cnode002 ....... [  DOWN  ] (Unassigned)
eu-west-1-cnode003 ....... [  DOWN  ] (Unassigned)
eu-west-1-director ....... [  DOWN  ]
node001 .................. [  DOWN  ]
node002 .................. [  DOWN  ]
basecm11 ................. [   UP   ]
```

The power control for the regular nodes must be functioning properly, or otherwise `cm-scale` cannot power them on and off.

If the head node has the `slurmserver` role, and the regular nodes have the `slurmclient` role, then

the roleoverview command should show something like:

Example

```
[basecm11->device[basecm11]]% roleoverview
Role              Nodes                                 Categories                    Nodes up
----------------- ------------------------------------- ----------------------------  --------
boot              basecm11                                                            1 of 1
cgroupsupervisor  eu-west-1-cnode001..eu-west-1-cnode002 aws-cloud-director,default    1 of 6
                  ,eu-west-1-director,node001..node002   ,eu-west-1-cloud-node
                  ,basecm11
clouddirector     eu-west-1-director                                                  0 of 1
cloudgateway      basecm11                                                            1 of 1
login             basecm11                                                            1 of 1
master            basecm11                                                            1 of 1
monitoring        basecm11                                                            1 of 1
provisioning      eu-west-1-director,basecm11                                         1 of 2
slurmclient       eu-west-1-cnode001..eu-west-1-cnode002 default,eu-west-1-cloud-node  0 of 3
                  ,node001..node002
slurmserver       basecm11                                                            1 of 1
storage           eu-west-1-director,basecm11            aws-cloud-director           1 of 2
```

A test user, fred can be created by the administrator (section 6.2), and an MPI hello executable based on the hello.c code (from section 3.5.1 of the *User Manual*) can be built:

Example

```
[fred@basecm11 ~]$ module add shared openmpi/gcc/64 slurm
[fred@basecm11 ~]$ mpicc hello.c -o hello
```

A batch file slurmhello.sh (from section 5.3.1 of the *User Manual*) can be set up. Restricting it to 1 process per node so that it spreads over nodes easier for the purposes of the test can be done with the settings:

Example

```
[fred@basecm11 ~]$ cat slurmhello.sh
#!/bin/sh
#SBATCH -o my.stdout
#SBATCH --time=30 #time limit to batch job
#SBATCH --ntasks=1
#SBATCH --ntasks-per-node=1
module add shared openmpi/gcc/64 slurm

mpirun /home/fred/hello
```

A default cluster can queue or run jobs without cm-scale being used. The default behavior is modified in the next steps, which bring in the cm-scale service:

The administrator assigns the ScaleServer role to the head node.

Example

```
[root@basecm11 ~]# cmsh
[basecm11]% device roles master
[basecm11->device[basecm11]->roles]% assign scaleserver
```

Within the assigned scaleserver role, a new dynamic node provider can be set, and properties for the dynamic pool of nodes can be set for the cloud compute nodes. Here the properties that are set are priority (page 483), templatenode (page 484), noderange (page 484), and extranodes (page 485).

Example

```
[basecm11->device*[basecm11*]->roles*[scaleserver*]]% resourceproviders
...->roles[scaleserver]->resourceproviders]% add dynamic pool2
...resourceproviders*[pool2*]]% set priority 2
...resourceproviders*[pool2*]]% set noderange eu-west-1-cnode001..eu-west-1-cnode002
...resourceproviders*[pool2*]]% set templatenode eu-west-1-cnode001
...resourceproviders*[pool2*]]% set extranodes eu-west-1-director
...resourceproviders*[pool2*]]% commit
...resourceproviders[pool2]]%
```

The regular compute nodes, `node001..node002` should be specified as nodes in the static pool.

The administrator may notice the similarity of dynamic and static pool configuration. The BCM front end has deliberately been set up to present dynamic pool and static pool nodes to the cluster administrator as two different configuration methods. This is because separating the pool types as dynamic and static pools is simpler for the cluster administrator to deal with. This way, regular compute nodes are treated, not as a special case of a dynamic pool, but simply as static pool nodes. The fundamental reason behind this separate treatment is because physical nodes cannot "materialize" dynamically with properties in the way the cloud compute nodes–which are virtualized nodes—can, due to the need to associate a MAC address with a physical node.

Assigning regular compute nodes to a static pool can be done in a similar way to what was shown before in the example on page 498.

Continuing with the current session, the nodes `node001..node002` are added to the static pool of nodes, `on-premises-nodes`. For this example they are set to a lower priority than the cloud nodes:

Example

```
...->roles[scaleserver]->resourceproviders]% add static on-premises-nodes
...->roles*[scaleserver*]->resourceproviders*[on-premises-nodes*]]% set nodes node001..node002
...->roles*[scaleserver*]->resourceproviders*[on-premises-nodes*]]% set priority 1
...->roles*[scaleserver*]->resourceproviders*[on-premises-nodes*]]% commit
...->roles[scaleserver]->resourceproviders[on-premises-nodes]]%
```

What this lower priority means is that a node that is not up and is in the static pool of nodes, is only powered on after all the cloud nodes are powered on and busy running jobs. If there happen to be nodes from the static pool that are already up, but are not running jobs, then these nodes take a job, despite the lower priority of the static pool, and irrespective of whether the dynamic pool nodes are in use.

Job priorities can be overridden in `cm-scale` by:

- allowing locations by setting `Mix Locations` to `true` (page 508) or
- pinning queues by setting `Pin Queues` to `true` (page 510)

A Slurm engine, and queue tracker for the `defq` are set as follows:

Example

```
...]->roles[scaleserver]]% engines
...]->roles[scaleserver]->engines]% add hpc slurm2
...*]->roles*[scaleserver*]->engines*[slurm2]]% set wlmcluster slurm
...*]->roles*[scaleserver*]->engines*[slurm2]]% trackers
...*]->roles*[scaleserver*]->engines*[slurm2]->trackers]% add queue tr2
...*]->roles*[scaleserver*]->engines*[slurm2]->trackers*[tr2*]]% set queue defq
...*]->roles*[scaleserver*]->engines*[slurm2*]->trackers*[tr2*]]% commit
...->roles[scaleserver]->engines[slurm2]->trackers[tr2]]%
```

The nodes node001..node002 and eu-west-1-cnode001..eu-west-1-cnode002 should already be in the queue defq by default, ready to run the jobs:

Example

```
...->roles[scaleserver]->engines[slurm2]->trackers[tr2]]% wlm use slurm; jobqueue; get defq nodes
eu-west-1-cnode001
eu-west-1-cnode002
node001
node002
```

The roleoverview (page 501) command is also handy for an overview, and to confirm that the role assignment of these nodes are all set to the SlurmClient role:

With these settings, the nodes in the dynamic pool can now be powered up or down on demand by cm-scale service, depending on the number of jobs that are pending. When the new ScaleServer role is committed in cmsh or Base View, then the cm-scale is run periodically. Each time it is run, cm-scale checks whether the node states should be changed. If needed, the administrator can check the log file /var/log/cm-scale to see what the service is doing.

Job submission can now be carried out, and the scaleserver assignment carried out earlier scales the cluster to cope with jobs according to the configuration that has been carried out in the session.

Before submitting the batch jobs, the administrator or user can check the jobs that are queued and running with the squeue command. If there are no jobs yet submitted, the output is simply the squeue headers, with no job IDs listed:

Example

```
[fred@basecm11 ~]$ squeue
JOBID PARTITION     NAME     USER ST       TIME  NODES NODELIST(REASON)
```

As in the previous example for the static pool only case (page 497), a way for user fred to flood the default queue defq is to run the batch file in a loop:

Example

```
[fred@basecm11 ~]$ while (true); do sbatch slurmhello.sh; done
Submitted batch job 1
Submitted batch job 2
Submitted batch job 3
...
```

After putting enough jobs into the queue (a few thousand should be enough, not more than five thousand would be sensible), the flooding can be stopped with a ctrl-c.

The changes in the queue can be watched by user fred:

Example

```
[fred@basecm11 ~]$ watch "squeue | head -5 ; squeue | tail -4"
Every 2.0s: squeue | head -5 ; squeue | tail -4              Wed Nov 22 16:08:52 2017

JOBID PARTITION     NAME     USER ST       TIME  NODES NODELIST(REASON)
    1      defq slurmhel     fred PD       0:00      1 (Resources)
    2      defq slurmhel     fred PD       0:00      1 (Resources)
    3      defq slurmhel     fred PD       0:00      1 (Resources)
    4      defq slurmhel     fred PD       0:00      1 (Resources)
 3556      defq slurmhel     fred PD       0:00      1 (Resources)
 3557      defq slurmhel     fred PD       0:00      1 (Resources)
 3558      defq slurmhel     fred PD       0:00      1 (Resources)
 3559      defq slurmhel     fred PD       0:00      1 (Resources)
```

The `head -4` and `tail -4` filters here are convenient for showing just the first 4 rows and last 4 rows of the very long `squeue` output, and skipping the bulk of the queue.

The preceding output illustrates how, with the jobs queued up, nothing is being processed yet from jobs number 1 to 3559 due to the resources not yet being available.

At this point `cm-scale` should have noticed that jobs are queued and that resources are needed to handle the jobs.

It should be noted that, at the time of writing of this section (January 2023), Slurm job processing with Auto Scaler currently only works as expected if `sbatch` rather than `srun` is used for dynamic jobs. The reason behind this `srun` quirk is explained on page 988.

At the start of this example session the cloud director is not up. So, `cm-scale` powers it up. This can be seen by running the `ds` command, or from CMDaemon info messages:

```
[basecm11->device]% ds | grep director
eu-west-1-director [  DOWN  ]
 then some time later:
eu-west-1-director [ PENDING ] (External ip assigned: 34.249.166.63, setting up tunnel)
 then some time later:
eu-west-1-director [     INSTALLING      ] (node installer started)
 then some time later:
eu-west-1-director [ INSTALLER_CALLINGINIT ] (switching to local root)
 then some time later:
eu-west-1-director [   UP   ]
```

If the cloud director is yet to be provisioned to the cloud from the head node for the very first time ("from scratch"), then that can take a while. Then, because the cloud compute nodes are in turn provisioned from the cloud director, it takes a while for the cloud compute nodes to be ready to run the jobs. So, the jobs just have to wait around in the queue until the cloud compute nodes are ready, before they are handled. Fortunately, the startup of a cloud director is by default much faster after the very first time.

A quick aside about how provisioning is speeded up the next time around: The cloud compute nodes will be stopped if they are idle, and after there are no more jobs in the queue, because the jobs have all been dealt with. Then, when the `extranodeidletime` setting has been exceeded, the cloud director is also stopped. The next time that jobs are queued up, all the cloud nodes are provisioned from a stopped state, rather than from scratch, and so they are ready for job execution much faster. Therefore, unlike the first time, the jobs queued up the next time are processed with less waiting around.

Getting back to how things proceed in the example session after the cloud director is up: `cm-scale` then provisions the cloud compute nodes `eu-west-1-node001` and `eu-west-1-node002` from the cloud director.

Example

```
[basecm11->device]% ds | grep cnode
eu-west-1-cnode001 ....... [ PENDING ] (Waiting for instance to start)
eu-west-1-cnode002 ....... [ PENDING ] (Waiting for instance to start)
 then some time later:
eu-west-1-cnode002 [     INSTALLING      ] (node installer started)
eu-west-1-cnode001 [     INSTALLING      ] (node installer started)
 and so on
```

Once these cloud compute nodes reach the state of `UP`, they can start to process the jobs in the queue. The queue activity then would show something like:

Example

when the dynamic pool nodes are being readied for job execution:

```
[fred@basecm11 ~]$ squeue | head -5 ; squeue | tail -4
JOBID PARTITION     NAME     USER ST       TIME  NODES NODELIST(REASON)
    1      defq slurmhel     fred PD       0:00      1 (Resources)
    2      defq slurmhel     fred PD       0:00      1 (Resources)
    3      defq slurmhel     fred PD       0:00      1 (Resources)
    4      defq slurmhel     fred PD       0:00      1 (Resources)
 3556      defq slurmhel     fred PD       0:00      1 (Resources)
 3557      defq slurmhel     fred PD       0:00      1 (Resources)
 3558      defq slurmhel     fred PD       0:00      1 (Resources)
 3559      defq slurmhel     fred PD       0:00      1 (Resources)
```

then later:

```
JOBID PARTITION     NAME     USER ST       TIME  NODES NODELIST(REASON)
   11      defq slurmhel     fred CF       0:00      1 eu-west-1-cnode001
   12      defq slurmhel     fred CF       0:00      1 eu-west-1-cnode002
   13      defq slurmhel     fred CF       0:00      1 (priority)
   14      defq slurmhel     fred CG       0:00      1 (priority)
 3556      defq slurmhel     fred PD       0:00      1 (Priority)
 3557      defq slurmhel     fred PD       0:00      1 (Priority)
 3558      defq slurmhel     fred PD       0:00      1 (Priority)
 3559      defq slurmhel     fred PD       0:00      1 (Priority)
```

then later, when `cm-scale` *sees all of the dynamic pool is used up, the lower priority static pool gets started up:*

```
JOBID PARTITION     NAME     USER ST       TIME  NODES NODELIST(REASON)
  165      defq slurmhel     fred CF       0:00      1 eu-west-1-cnode001
  166      defq slurmhel     fred CF       0:00      1 node001
  168      defq slurmhel     fred CG       0:00      1 node002
 3556      defq slurmhel     fred PD       0:00      1 (Priority)
 3557      defq slurmhel     fred PD       0:00      1 (Priority)
 3558      defq slurmhel     fred PD       0:00      1 (Priority)
 3559      defq slurmhel     fred PD       0:00      1 (Priority)
  167      defq slurmhel     fred  R       0:00      1 eu-west-1-cnode002
```

In `cmsh`, the priority can be checked with:

Example

```
[basecm11 ->device[basecm11]->roles[scaleserver]->resourceproviders]% list
Name (key)        Priority     Enabled
----------------- ------------ -------
on-premises-nodes 1            yes
pool2             2            yes
```

Also in `cmsh`, the jobs can be listed via the `jobs` submode:

Example

```
[basecm11->wlm[slurm]->jobs]% list | head -5 ; list | tail -4
Type   Job ID User  Queue Running time Status       Nodes
------ ------ ----- ----- ------------ ----------   ------------------
Slurm  334    fred  defq  1s           COMPLETED    eu-west-1-cnode001
Slurm  336    fred  defq  1s           COMPLETED    node001
Slurm  3556   fred  defq  0s           PENDING
Slurm  3557   fred  defq  0s           PENDING
```

```
Slurm  3558   fred  defq  0s          PENDING
Slurm  3559   fred  defq  0s          PENDING
Slurm  335    fred  defq  1s          RUNNING      eu-west-1-cnode002
[basecm11->wlm[slurm]->jobs]%
```

Eventually, when the queue has been fully processed, the jobs are all gone:

Example

```
[fred@basecm11 ~]$ squeue
JOBID PARTITION     NAME     USER ST       TIME  NODES NODELIST(REASON)
```

With the current configuration the cloud compute nodes in the dynamic pool `pool2` are powered up before the regular compute nodes in the static pool `on-premises-nodes`. That is because the cloud compute nodes have been set by the administrator in this example to have a higher priority. This is typically sub-optimal, and is actually configured this way just for illustrative purposes. In a real production cluster, the priority of regular nodes is typically going to be set higher than that for cloud compute nodes, because using on-premises nodes is likely to be cheaper.

The administrator can also check on the job status via the job metrics of `cmsh` too, using the options to the `filter` command, such as `--pending` or `--running`:

Initially, before the jobs are being run, something like this will show up:

Example

```
[root@basecm11 ~]# cmsh -c "wlm use slurm; jobs ; watch filter --running -u fred"
Every 2.0s: filter --running -u fred                          Wed Nov 22 16:03:18 2017
No jobs found
```

Then, eventually, when the jobs are being run, the cloud nodes, which have a higher priority, start job execution, so that the output looks like:

Example

```
Every 2.0s: filter --running -u fred                          Wed Nov 22 16:50:35 2017
Job ID Job name      User Queue Submit time Start time End time Nodes           Exit code
------ ------------- ---- ----- ----------- ---------- -------- --------------- ---------
406    slurmhello.sh fred defq  16:16:56    16:27:21   N/A      eu-west1-cnode001 0
407    slurmhello.sh fred defq  16:16:56    16:27:21   N/A      eu-west1-cnode002 0
```

and eventually the regular on-site nodes which are originally down are started up by the ScaleServer and are also listed.

8.4 Further `cm-scale` Configuration And Examples

8.4.1 Dynamic Nodes Re-purposing

Sometimes it is useful to share the same nodes among several queues, and reuse the nodes for jobs from other queues. This can be done by dynamically assigning node categories in `cm-scale`. Different settings, or a different software image, then run on the re-assigned node after re-provisioning.

The feature is enabled by setting `Assign Category` parameter in the tracker settings.

For example, the following case uses Slurm as the workload engine, and sets up two queues `chem_q` and `phys_q`. Assuming in this example that jobs that are to go to `chem_q` require chemistry software on the node, but jobs for `phys_q` require physics software on the node, and that for some reason the softwares cannot run on the node at the same time. Then, the nodes can be re-purposed dynamically. That is, the same node can be used for chemistry or physics jobs by setting up the appropriate configuration for it. In this case the same node can be used by jobs that require a different configuration, software, or even operating system. The trackers configuration may then look as follows:

Example

```
[basecm11->device[basecm11]->roles[scaleserver]->engines[slurm]->trackers[chem]]% show
Parameter                      Value
------------------------------ ------------------------------------------------
Type                           ScaleHpcQueueTracker
Name                           chem
Queue                          chem_q
Enabled                        yes
Assign Category                chem_cat
Primary Overlays
[basecm11->device[basecm11]->roles[scaleserver]->engines[slurm]->trackers[chem]]% use phys
[basecm11->device[basecm11]->roles[scaleserver]->engines[slurm]->trackers[phys]]% show
Parameter                      Value
------------------------------ ------------------------------------------------
Type                           ScaleHpcQueueTracker
Name                           chem
Queue                          chem_q
Enabled                        yes
Assign Category                phys_cat
Primary Overlays
[basecm11->device[basecm11]->roles[scaleserver]->engines[slurm]->trackers[phys]]%
```

Assuming that initially there are two nodes, node001 and node002, both in category chem_cat. Then, when cm-scale finds a pending job in queue phys_q, it may decide to assign category phys_cat to either node001, or to node002. In this way the number of nodes serving queue phys_q increases and number of nodes serving chem_q decreases, in order to handle the current workload. When the job is finished, the old node category is not assigned back to the node, until a new job appears in chem_q and requires this node to have the old category.

8.4.2 Pending Reasons

This section is related only to HPC engines (workload managers). In this section, the term job is used instead of workload.

If cm-scale makes a decision on how many nodes should be started for a job, then it checks the status of the job first. If the job status is pending, then it checks the list of *pending reasons* for that job. The checks are to find pending reasons that prevent the job from starting when more free nodes become available.

A pending reason can be one of the following 3 types:

Type 1: allows a job to start when new free nodes become available

Type 2: prevents a job from starting on particular nodes only

Type 3: prevents a job from starting anywhere

Each pending reason has a text associated with it. The text is usually printed by the workload manager job statistics utilities. The list of pending reasons texts of types 1 and 2 can be found in the pending reasons exclude file, /cm/local/apps/cm-scale/lib/python3.12/site-packages/cmscale/trackers/hpc_queue/pending_reasons/WLM.exclude, where WLM is a name of workload manager specified in the configuration of the engine in ScaleServer role.

In the pending reasons exclude file, the pending reason texts are listed as one reason per line. The reasons are grouped in two sublists, with headers:

- [IGNORE_ALWAYS]
- [IGNORE_NO_NODE]

The [IGNORE_ALWAYS] sublist lists the type 1 pending reason texts. If a job has only this group of reasons, then cm-scale considers the job as ready to start, and attempts to create or boot compute nodes for it.

The [IGNORE_NO_NODE] sublist lists the type 2 pending reason texts. If the reason does not specify the hostname of a new free node at the end of a pending reason after the colon (":"), then the job can start on the node. If the reason does specify the hostname of a new free node after the colon, and if the hostname is owned by one of the managed nodes—nodes that can be stopped/started/created by cm-scale—then the job is considered as one that is not to start, when nodes become available.

If a job has a pending reason text that is not in the pending reasons exclude file, then it is assumed to be a type 3 reason. New free nodes for such a job do not get the job started.

If there are several pending reason texts for a job, then cm-scale checks all the pending reasons one by one. If all reasons are from the IGNORE_ALWAYS or IGNORE_NO_NODE sublists, and if a pending reason text matched in the IGNORE_NO_NODE sublist does not include hostnames for the managed nodes, only then will the job be considered as one that can be started just with new nodes.

Custom Pending Reasons

If the workload manager supports them, then custom pending reason texts are also supported. The administrator can add a pending reason text to one of the sections in the pending reasons exclude file.

The cm-scale service checks only if the pending reason text for the job starts with a text from the pending reasons file. It is therefore enough to specify just a part of the text of the reason in order to make cm-scale take it into account. Regular expressions are also supported. For example, the next two pending reason expressions are equivalent when used to match the pending reason text `Not enough job slot(s):`

Example

- `Not enough`
- `Not enough [a-z]* slot(s)`

The workload manager statistics utility can be used to find out what custom pending reason texts there are, and to add them to the pending reasons file. To do this, some test job can be forced to have such a pending reason, and the output of the job statistics utility can then be copy-pasted. For example, LSF shows custom pending reasons that look like this:

Example

```
Customized pending reason number <integer>
```

Here, <*integer*> is an identifier (an unsigned integer) for the pending reason, as defined by the administrator.

8.4.3 Locations

Sometimes it makes sense to restrict the workload manager to run jobs only on a defined subset of nodes. For example, if a user submits a multi-node job, then it is typically better to run all the job processes either on the on-premises nodes, or on the cloud nodes. That is, without mixing the node types used for the job. The *locations* feature of cm-scale allows this kind of restriction for HPC workload managers.

The cm-scale configuration allows one of these two modes to be selected:

1. *forced location*: when the workload is forced to use one of the locations chosen by cm-scale,
2. *unforced location*: when workloads are free to run on any of the compute nodes that are already managed (running, freed or started) by cm-scale. This is the default if Auto Scaler is set up.

In NVIDIA Base Command Manager 11, for a forced location, cm-scale supports these two different locations:

1. local: on-premises nodes,
2. cloud: AWS instances (Chapter 3 of the *Cloudbursting Manual*) or Azure instances (Chapter 5 of the *Cloudbursting Manual*)

To restrict the WLM location—that is to choose a forced location—the mixlocations advanced setting in the scaleserver role for the node must be set to no

Example

```
[basecm11->device[basecm11]->roles[scaleserver]->advancedsettings]% set mixlocations no
[basecm11->device*[basecm11*]->roles*[scaleserver*]->advancedsettings*]% commit
```

The location is automatically configured by BCM when the node is added to the workload manager. Details per workload manager are described next.

Slurm

Slurm does not allow the assignment of node properties—features, in Slurm terminology—to jobs if no node exists that is labeled by this property. Thus any property used must be added to some node. This can be the template node if a dynamic resource provider is used, or it can be an appropriate off-premises node if a static resource provider is used. If the slurmclient role is assigned to a node—for example, a template node—then the location value for this node is automatically configured by BCM.

The current location value can be found using the scontrol command. For example, for node001:

Example

```
[root@basecm11 ~]# module load slurm
[root@basecm11 ~]# scontrol show node node001 | grep AvailableFeatures
```

LSF

In order to allow cm-scale to restrict LSF jobs, BCM configures a generic resource called location per node. The resource is added as a string resource in lsf.cluster.<*CLUSTER_NAME:*> configuration file.

The location value for this node is automatically configured by BCM.

To verify that the resource is added, the lshosts -s command can be run:

Example

```
[root@basecm11 ~]# lshosts -s location | head -1;           lshosts -s location | grep node001
RESOURCE                              VALUE       LOCATION
location                              local       node001.cm.cluster
```

8.4.4 Azure Storage Accounts Assignment

If an Azure node is cloned manually from some node or node template, then the Azure node gets the same storage account as the node it has been cloned from. This may slow the nodes down if too many nodes use the same storage account. The cm-scale utility can therefore assign different storage accounts to nodes that are cloned like this.

The maximum number of nodes for such a storage account is defined by the AZURE_DISK_ACCOUNT_NODES parameter. This parameter has a value of 20 by default, and can be changed in the configuration file /cm/local/apps/cm-scale/lib/python3.12/site-packages/cmscale/config.py. The cm-scale utility must be restarted after the change.

The newly-cloned-by-cm-scale Azure node gets a randomly-generated storage account name if other storage accounts already have enough nodes associated with them. That is, if other storage accounts have AZURE_DISK_ACCOUNT_NODES or more nodes.

The storage account name is assigned in the node cloud settings in storage submode. For example, in cmsh, the assigned storage accounts can be viewed as follows:

Example

```
[root@basecm11 ~]# cmsh
[basecm11]% device use cnode001
[basecm11->device[cnode001]]% cloudsettings
[basecm11->device[cnode001]->cloudsettings]% storage
[basecm11->...[cnode001]->cloudsettings->storage]% get root-disk storageaccountname
azurepaclogzjus1
[basecm11->...[cnode001]->cloudsettings->storage]% get node-installer-disk storageaccountname
azurepaclogzjus1
[basecm11->device[cnode001]->cloudsettings->storage]% ..
[basecm11->device[cnode001]->cloudsettings]% get bootdiagnosticsstorageaccountname
azurepaclogzjus1
[basecm11->device[cnode001]->cloudsettings]%
```

If a node is terminated and removed from the BCM configuration, then the storage account remains in Azure. It has to be explicitly manually removed by the administrator.

8.4.5 Uptake of HPC Jobs By Particular Types Of Nodes

By default, `cm-scale` assumes that an HPC job submitted to a particular queue can take a node from outside the queue. This is because by assigning a category, or moving the node to a configuration overlay, the node will be moved to the appropriate queue eventually. From this point of view, the nodes form a single resource pool, and the nodes in the pool are re-purposed on demand.

In some scenarios there is a need for certain types of HPC jobs run only on particular types of nodes, without the nodes being re-purposed. A typical example: jobs with GPU code require cloud nodes that have access to GPU accelerators, while jobs that do not have GPU code can use the less expensive non-GPU cloud nodes. For this case then, the GPU cloud node is started when the GPU job requires a node, and otherwise a non-GPU node is started.

Job segregation is achieved in `cm-scale` as follows:

1. The `Pin Queues` setting, which is an advanced setting in the `scaleserver` role for the node, is enabled:

   ```
   [basecm11->device[basecm11]->roles[scaleserver]->advancedsettings]% set pinqueues yes
   [basecm11->device*[basecm11*]->roles*[scaleserver*]->advancedsettings*]% commit
   ```

2. A new queue is created, or an existing one is used. The queue is used for the jobs that require a particular node type.

3. The particular node type is added to this queue. If the node is already defined in BCM, then the administrator can assign the queue to the node in the workload manager client role. For example, if the workload manager is Slurm, then the queue is assigned to the nodes in the `slurmclient` role. If the node has not been defined yet and will be cloned on demand (according to the dynamic resource provider settings, page 484), then its template node is assigned to the queue. When a new node is cloned from the template, the queue is then inherited from the template node.

4. The previous two steps are repeated for each job type.

After that, if a user submits a job to one of the queues, then `cm-scale` starts or clones a node that is linked with the job queue.

The following `cmsh` session snippet shows a configuration example:

Example

```
[root@basecm11 ~]# cmsh
[basecm11]% device roles master
[basecm11->device[basecm11]->roles]% use scaleserver
[basecm11->...roles[scaleserver]]% resourceproviders
[basecm11->...roles[scaleserver]->resourceproviders]% add dynamic rp1
[basecm11->...roles[scaleserver]->resourceproviders*[rp1*]]% set templatenode tnode1
[basecm11->...roles[scaleserver]->resourceproviders*[rp1*]]% set noderange cnode001..cnode100
[basecm11->...roles[scaleserver]->resourceproviders*[rp1*]]% commit
[basecm11->...roles[scaleserver]->resourceproviders[rp1]]% clone rp2
[basecm11->...roles[scaleserver]->resourceproviders*[rp2*]]% set templatenode tnode2
[basecm11->...roles[scaleserver]->resourceproviders*[rp2*]]% set noderange cnode101..cnode200
[basecm11->...roles[scaleserver]->resourceproviders*[rp2*]]% commit
[basecm11->...roles[scaleserver]->resourceproviders[rp2]]% ..;..
[basecm11->...roles[scaleserver]]% engines
[basecm11->...roles[scaleserver]->engines]% add hpc s1
[basecm11->...roles[scaleserver]->engines*[e1*]]% set workloadmanager slurm
[basecm11->...roles[scaleserver]->engines*[e1*]]% trackers
[basecm11->...roles[scaleserver]->engines*[e1*]->trackers]]% add queue tr1
[basecm11->...roles[scaleserver]->engines*[e1*]->trackers*[tr1*]]% set queue q1
[basecm11->...roles[scaleserver]->engines*[e1*]->trackers*[tr1*]]% commit
[basecm11->...roles[scaleserver]->engines[e1]->trackers[tr1]]% clone tr2
[basecm11->...roles[scaleserver]->engines*[e1*]->trackers*[tr2*]]% set queue q2
[basecm11->...roles[scaleserver]->engines*[e1*]->trackers*[tr2*]]% commit
[basecm11->...roles[scaleserver]->engines[e1]->trackers[tr2]]% category
[basecm11->category]% clone default cat1
[basecm11->category*[cat1*]]% roles
[basecm11->category*[cat1*]->roles*]% assign slurmclient
[basecm11->category*[cat1*]->roles*[slurmclient*]]% set queues q1
[basecm11->category*[cat1*]->roles*[slurmclient*]]% commit
[basecm11->category[cat1]->roles[slurmclient]]% category clone cat1 cat2
[basecm11->category*[cat2*]->roles*[slurmclient*]]% set queues q2
[basecm11->category*[cat2*]->roles*[slurmclient*]]% commit
[basecm11->category[cat2]->roles[slurmclient]]% device use tnode1
[basecm11->device[tnode1]]% set category cat1
[basecm11->device*[tnode1*]]% commit
[basecm11->device[tnode1]]% device use tnode2
[basecm11->device[tnode2]]% set category cat2
[basecm11->device*[tnode2*]]% commit
[basecm11->device[tnode2]]%
```

Using the preceding configuration, the user may submit a job with a regular workload manager submission utility specifying the queue q1 or q2, depending on whether the job requires nodes that should be cloned from tnode1 or from tnode2.

8.4.6 How To Exclude Unused Nodes From Being Stopped

If a node is idle, then by default cm-scale automatically stops or terminates the node.

However, in some cases there may be a need to start a node on demand, and when it becomes idle, there may be a need to keep the node running. This can be useful if the administrator would like to investigate the performance of an application, or to debug some issues. After completing the investigation or debug session, the administrator can stop the node manually.

The parameter KEEP_RUNNING_RANGES keeps such nodes from being stopped or terminated. The parameter should be added to the configuration file /cm/local/apps/cm-scale/lib/python3.12/site-packages/cmscale/config.py. To have the changed setting take effect, the cm-scale service must be restarted.

KEEP_RUNNING_RANGES defines a map of resource provider names to node name ranges.

Extra nodes can be added to the range of the nodes. However, if the extra node must not be stopped or terminated by `cm-scale`, then for each resource provider that has such an extra node, the value of `extranodestop` must be set to `yes`.

In the following example, nodes `cnode002`, `cnode003`, `cnode004`, and `cnode010`, are associated with the `azurenodes1` resource provider. They are therefore never stopped or terminated by `cm-scale`. They are only started on demand by `cm-scale`.

The nodes `cnode012` and `cnode014` are associated with the `azurenodes2` resource provider. They are therefore also not stopped or terminated by `cm-scale`.

Example

```
opts = {
  [...]
  "KEEP_RUNNING_RANGES": {
    "azurenodes1": "cnode002..cnode004,cnode010",
    "azurenodes2": "cnode012,cnode014"
  }
}
```

8.4.7 Prolog And Epilog Scripts With Auto Scaler

Sometimes the administrator would like some actions to be performed for a workload when the Auto Scaler allocates and starts using a node, or when the Auto Scaler deallocates and stops using a node. The administrator can arrange such actions by configuring prolog and epilog scripts (section 7.3.4) in the resource provider. The scripts are then executed on the nodes running the Auto Scaler service, i.e. with the `ScaleServer` role.

Both the dynamic and the static resource providers (section 8.2.2) support the following options:

1. `allocationProlog`: path to a shell script that is executed just before a node is started up by Auto Scaler
2. `allocationEpilog`: path to a shell script that is executed just before a node is powered off by Auto Scaler
3. `allocationScriptsTimeout`: the prolog and epilog scripts timeout (the script that is running is killed if the timeout is exceeded).

The prolog script runs when an existing node is about to start, and also runs when a node has just been cloned and is also about start.

The epilog script runs when a node is stopped or when a cloud node is terminated.

The prolog and epilog scripts are run per node, and can run in parallel. Thus if synchronization between them is needed, then it should be implemented by the scripts themselves.

The standard output and error messages of the executed scripts are mixed and added to the Auto Scaler as `debug2` log messages (the `debug2` logs can be enabled in the `AdvancedSettings` submode of the ScaleServer role). It therefore makes sense to keep the output reasonably small, informative, and human readable.

When the scripts are run, Auto Scaler passes environment variables that can be used inside the scripts in order to decide what to do. These environment variables are:

1. `AS_NODE`: node short hostname which the script started for;
2. `AS_SCRIPT_TYPE`: either "epilog" or "prolog";
3. `AS_RESOURCE_PROVIDER`: name of the resource provider where this script is configured;
4. `AS_ENGINE`: workload engine name, which workload requires the node ("unknown" if no workload requires the node).

By default the scripts are not defined, and therefore nothing is executed by default when nodes are stopped, terminated or started.

8.4.8 Queue Node Placeholders

A queue node placeholder is a node that does not yet exist, but has a corresponding object that exists, and the object has queues defined, amongst other properties. It can be used to plan resource use.

Job Rejection For Exceeding Total Cluster Resources

At the time of job submission, the workload manager checks the total available number of slots (used and unused) in a queue. This is the sum of the available slots (used and unused) provided by each node in that queue.

- Jobs that require less than the total number of slots are normally made to wait until more slots become available.
- Jobs that require more than this total number of slots are normally rejected outright by the workload manager, without being put into a wait state. This is because workload managers normally follow a logic that relies on the assumption that if the job demands more slots than can exist on the cluster as it is configured at present, then the cluster will never have enough slots to allow a job to run.

Assuming The Resources Can Never Be Provided

The latter assumption, that a cluster will never have enough slots to allow a job to run, is not true when the number of slots is dynamic, as is the case when `cm-scale` is used. When `cm-scale` starts up nodes, it adds them to a job queue, and the workload manager is automatically configured to allow users to submit jobs to the enlarged queue. That is, the newly available slots are configured as soon as possible so that waiting jobs are dealt with as soon as possible. For jobs that have already been rejected, and are not waiting, this is irrelevant, and users would have to submit the jobs once again.

Ideally, in this case, the workload manager should be configured to know about the number of nodes and slots that can be started up in the future, even if they do not exist yet. Based on that, jobs that would normally be rejected, could then also get told to wait until the resources are available, if it turns out that configured future resources will be enough to run the job.

Slurm Resources Planning With Placeholders

Slurm allows nodes that do not exist yet to be defined. These are nodes with hostnames that do not resolve, and have the Slurm setting of `state=CLOUD` for cloud nodes, and `state=FUTURE` for other nodes. BCM allows Slurm to add such "fake" nodes to Slurm queues dynamically, when not enough real nodes have yet been added. BCM supports this feature only for Slurm at present.

This feature is not yet implemented for the other workload managers because they require the hostname of nodes that have been added to the workload manager configuration to be resolved.

Within the Slurm WLM instance it is possible to set a list of placeholder objects. In `cmsh` this can be done within the main `wlm` mode, selecting the Slurm instance, and then going into the `placeholders` submode. Each placeholder allows the following values to be set:

- `queue`: the queue name, used as key
- `maxnodes`: the maximum number of nodes that this queue allows
- `basenodename`: the base node name that is used when a new node name is generated
- `templatenode`: a template node that is used to provide user properties taken from its slurmclient role when new fake nodes are added.

For example, the following `cmsh` session uses the head node with an existing `slurm` instance to illustrate how the Slurm queue `defq` could be configured so that it always has a maximum of 32 nodes, with the nodes being like node001:

Example

```
[root@basecm11 ~]# scontrol show part defq | grep " Nodes="
Nodes=node001
[root@basecm11 ~]# cmsh
[basecm11]% wlm use slurm
[basecm11->wlm[slurm]]% placeholders
[basecm11->wlm[slurm]->placeholders]% add defq
[basecm11->wlm*[slurm*]->placeholders*[defq*]]% set maxnodes 32
[basecm11->wlm*[slurm*]->placeholders*[defq*]]% set basenodename placeholder
[basecm11->wlm*[slurm*]->placeholders*[defq*]]% set templatenode node001
[basecm11->wlm*[slurm*]->placeholders*[defq*]]% commit
[basecm11->wlm[slurm]->placeholders[defq]]%
[root@basecm11 ~]# scontrol show part defq | grep " Nodes="
   Nodes=node001,placeholder[01-31]
```

If a new real node is added to the queue, then the number of placeholder nodes is decreased by one.

The placeholders can also be configured in Base View via the HPC resource, using the navigation path:

`HPC > Workload Management Clusters >` *Slurm instance* `> Placeholders`

Preventing `slurmctld` From Restarting

If the number of nodes is changed, or if their names are changed in `slurm.conf`, then CMDaemon restarts the Slurm server daemon, `slurmctld`, to apply the changes. If the administrator needs to prevent `slurmctld` from restarting each time that a new node is added to Slurm, then the nodes can be added, or cloned, to the BCM configuration manually, even if they do not have any IP address assigned yet. This is assuming that they get their IP addresses assigned over DHCP later on.

If the nodes are added to the configuration manually, then CMDaemon restarts `slurmctld` only once. This means that, when `cm-scale` starts the nodes, CMDaemon does not restart `slurmctld`.

8.4.9 Auto Scaling A Job On-premises To A Workload Manager And Kubernetes

In the session for this section, a cluster with 4 nodes is assumed. A workload manager such as Slurm is assumed to be already set as the engine (**Use Case: Workload Manager (On-premises)**, page 474).

If the cluster administrator now would also like to make a Kubernetes engine available to jobs, as suggested in the use case 2 on page 467, then it can be added within the `scaleserver` role as follows:

Example

```
[basecm11->configurationoverlay[autoscaler]->roles[scaleserver]->engines]% list
Name (key)       Priority
---------------- ------------
slurm            0
[basecm11->configurationoverlay...->engines]% add kubernetes k8s
[basecm11->configurationoverlay*...->engines*[k8s*]]% set cluster default
[basecm11->configurationoverlay*...->engines*[k8s*]]% trackers
[basecm11->configurationoverlay*...->engines*[k8s*]->trackers]% add namespace default
[basecm11->configurationoverlay*...[k8s*]->trackers*[default*]]% set controllernamespace default
[basecm11->configurationoverlay*...->engines*[k8s*]->trackers*[default*]]% commit
[basecm11->configurationoverlay...->engines[k8s]->trackers[default]]%
```

In the trackers for each engine, the overlay to move to must be specified:

Example

```
[basecm11->configurationoverlay...->engines[k8s]->trackers[default]]% set primaryoverlays kube-default-worker
[basecm11->configurationoverlay*...->engines*[k8s*]->trackers*[default*]]% commit
[basecm11->configurationoverlay...->engines[k8s]->trackers[default]]% ..
[basecm11->configurationoverlay...->engines[k8s]->trackers]% ..
[basecm11->configurationoverlay...->engines[k8s]]% ..
[basecm11->configurationoverlay...->engines]% use slurm
[basecm11->configurationoverlay...->engines[slurm]]% trackers
[basecm11->configurationoverlay...->engines[slurm]->trackers]% use defq
[basecm11->configurationoverlay...->engines[slurm]->trackers[defq]]% set primaryoverlays slurm-client
[basecm11->configurationoverlay*...->engines*[slurm*]->trackers*[defq*]]% commit
[basecm11->configurationoverlay...->engines[slurm]->trackers[defq]]%
```

Since the cluster is entirely on-premises, and no cloud nodes are to be used, there is no need to configure a dynamic provider (section 8.3.2).

To allow movement of jobs from one queue to another, queue pinning must be disabled:

Example

```
[basecm11->configurationoverlay[autoscaler]->roles[scaleserver]->advancedsettings]% set pinqueues no
[basecm11->configurationoverlay*[autoscaler*]->roles*[scaleserver*]->advancedsettings*]% commit
[basecm11->configurationoverlay[autoscaler]->roles[scaleserver]->advancedsettings]%
```

Auto scaler normally takes the amount of memory into account from the workload manager. However, Slurm does not know about the memory of nodes that are not managed by it. A default memory size should therefore be set for when a job requirement is matched to Slurm, using the default resources specification (page 470):

Example

```
[basecm11->configurationoverlay...->resourceproviders[static]]% set defaultresources "mem_free:slurm=7GB"
[basecm11->configurationoverlay*[autoscaler*]->roles*[scaleserver*]->resourceproviders*[static*]]% commit
[basecm11->configurationoverlay[autoscaler]->roles[scaleserver]->resourceproviders[static]]%
```

The nodes reboot when a job requires it:

Example

```
[basecm11->configurationoverlay[autoscaler]->roles[scaleserver]->resourceproviders[static]]%
Tue Aug 16 11:43:40 2022 [notice] basecm11: node003 [       BOOTING       ] (ldlinux.c32 from basecm11)
Tue Aug 16 11:43:40 2022 [notice] basecm11: node004 [       BOOTING       ] (ldlinux.c32 from basecm11)
[basecm11->configurationoverlay[autoscaler]->roles[scaleserver]->resourceproviders[static]]%
Tue Aug 16 11:44:22 2022 [notice] basecm11: node004 [     INSTALLING      ] (node installer started)
[basecm11->configurationoverlay[autoscaler]->roles[scaleserver]->resourceproviders[static]]%
Tue Aug 16 11:44:24 2022 [notice] basecm11: node003 [     INSTALLING      ] (node installer started)
[basecm11->configurationoverlay[autoscaler]->roles[scaleserver]->resourceproviders[static]]%
```

8.4.10 AWS Spot Instances And Availability Zones

Amazon cloud regions consist of multiple, isolated, *availability zones*. The number of spot instances that can be started in each zone is limited by the capacity of a zone. If the spot instances are configured to start in several different availability zones, then Auto Scaler detects this and tries to start nodes in those various different availability zones. At the start of each iteration of `cm-scale`, if `cm-scale` sees that the capacity of a zone is exhausted, then all nodes in that zone are considered to be unavailable for starting up by `cm-scale`.

Configuring several sets of spot instances in availability zones allows running nodes to be started up, even if the capacity in one or more zones is exhausted. For this configuration, the cluster administrator has two options:

1. Configuring the nodes from different availability zones in one workload manager job queue or Kubernetes namespace.
2. Having the nodes in different queues/namespaces, and letting users submit the jobs to multiple queues at the same time, so that `cm-scale` goes through the queues when selecting a node to start. This option requires multi-queue support in the workload manager. In this case, the `allowedresourceproviders` parameter should not be set within the `trackers` submode (section 8.2.6) for the engine.

If cloud nodes (spot instances) from different availability zones are added to the `scaleserver` role, then the lack of capacity is recognized by `cm-scale` automatically, and no additional configuration is needed.

The cluster administrator can however still override the availability zones information for `cm-scale`. This can be carried out by modifying the `cm-scale` configuration file:
`/cm/local/apps/cm-scale/lib/*/site-packages/cmscale/config.py`
In the file, a new parameter `AVAILABILITY_ZONES` must be added to the opts dictionary. The format the parameter takes is as follows:

Example

`{"AVAILABILITY_ZONES" : {`*<provider name>* `: {`*<availability zone name>* `:` *<node list in node range format>*`}}}`

For example, for the `us-west-*` availability zones:

Example

```
"AVAILABILITY_ZONES": {
    "aws": {
        "us-west-2a": "cpu-001..cpu-005, cpu-spot-a-001..cpu-spot-a-010",
        "us-west-2c": "cpu-spot-c-001..cpu-spot-c-010",
        "us-west-2d": "cpu-spot-d-001..cpu-spot-d-010",
    },
},
```

Currently, availability zones are supported by Auto Scaler only for AWS. Auto Scaler ignores any lack of capacity in the availability zones of other cloud providers.

8.4.11 Auto Scaler Statistics

Internal Auto Scaler metrics can be collected and visualized with BCM monitoring. The metrics can be used to help debug some issues, or can be used to analyze how Auto Scaler works over longer periods of time.

When statistics collection is enabled, Auto Scaler pushes its metrics to CMDaemon. The metric data values are then accessible using `cmsh` and Base View.

Statistics collection can be enabled

- by enabling the option during Auto Scaler setup with `cm-auto-scaler-setup`. The option can be enabled in the `Auto Scaler base options` screen (figure 8.3)
  or
- by setting the `collectstatistics` parameter within the advanced settings of the `scaleserver` role.

Example

```
[basecm11->configurationoverlay[autoscaler]->roles[scaleserver]->advancedsettings]% set collectstatistics yes
[basecm11->configurationoverlay*[autoscaler*]->roles*[scaleserver*]->advancedsettings*]% commit
[basecm11->configurationoverlay[autoscaler]->roles[scaleserver]->advancedsettings]%
```

During Auto Scaler setup with cm-auto-scaler-setup, the AutoScaler standalone monitoring entity is added. This is what is used for metrics aggregation and viewing:

Example

```
[basecm11->monitoring]% standalone
[basecm11->monitoring->standalone]% use autoscaler
[basecm11->monitoring->standalone[AutoScaler]]% latestmetricdata
Measurable                          Parameter    Type          Value      Age        State      Info
----------------------------------- ------------ ------------ ---------- ---------- ---------- ----------
no_resources_workloads                           Workloads     4          24.1s
pending_workloads                                Workload      4          24.1s
pre_iteration_down_nodes                         Nodes         16         24.1s
pre_iteration_completely_up_nodes                Nodes         16         24.1s
running_workloads                                Workload      2          24.1s
no_capacity_spot_requests           eu-west-2a   Nodes         2          24.1s
no_capacity_availability_zones                   Nodes         1          24.1s
[basecm11->monitoring->standalone[AutoScaler]]%
```

In Base View, statistics for the Auto Scaler metrics can be found, after collection has started, using the navigation path starting at the icon with the label Monitoring that is to be found in the top part of the Base View display:

Monitoring > AutoScaler

This location should not be confused with the location for the icon with the label Monitoring that is to be found in the left-hand navigation column, which is about setting how the monitoring for Auto Scaler is used.

Monitoring > Standalone Monitored Entities > AutoScaler

The statistics metrics are divided into three classes: Actions, Nodes and Workloads:

1. Actions:
   - assigned_categories: number of node category assignments performed by Auto Scaler
   - cloned_nodes: number of nodes cloned by Auto Scaler
   - terminated_nodes: number of nodes terminated by Auto Scaler
   - spot_requests_terminations: number of spot instance requests terminated by Auto Scaler
   - removed_nodes: number of nodes removed by Auto Scaler
   - moved_to_overlays_nodes: number of nodes moved to overlays by Auto Scaler
   - deleted_from_overlays_nodes: number of nodes deleted from overlays by Auto Scaler
   - drained_nodes: number of nodes drained by Auto Scaler
   - undrained_nodes: number of nodes undrained by Auto Scaler
   - started_nodes: number of nodes started (powered on) by Auto Scaler
   - stopped_nodes: number of nodes stopped (powered off) by Auto Scaler
   - shutdown_nodes: number of nodes shut down by Auto Scaler
   - executed_prologs: number of prologs executed by Auto Scaler
   - executed_epilogs: number of epilogs executed by Auto Scaler
   - constrained_workloads: number of workloads (jobs, pods, ...) constrained by Auto Scaler
2. Nodes:
   - pre_iteration_completely_up_nodes: number of nodes running (completely up) at the beginning of each iteration

- pre_iteration_down_nodes: number of nodes in state down at the beginning of each iteration
- pre_iteration_pending_nodes: number of pending nodes at the beginning of each iteration
- no_capacity_spot_requests: number of pending spot instance requests that could not be fulfilled due to the lack of capacity per availability zone
- no_capacity_availability_zones: number of availability zones that nodes could not start due to the lack of capacity

3. Workloads:

- not_allowed_workloads: number of workloads for which a start was not allowed
- no_resources_workloads: number of workloads for which a resources request could not be satisfied
- pending_workloads: number of pending workloads
- running_workloads: number of running workloads
- failed_workloads: number of failed workloads
- unknown_workloads: number of workloads in an unknown state

9

Post-installation Software Management

Introduction: What Is Post-installation Software Management?

After NVIDIA Base Command Manager has been installed, administrators are expected to manage and update the distribution software and the cluster software as updates are packaged and made available by the distribution and by the BCM software development team. Managing and updating means carrying out software management actions such as installation, removal, updating, version checking, and so on.

Security And Hardening

For security vulnerabilities in BCM packages, BCM support can be contacted directly (section 14.2). Vulnerabilities that are part of the underlying distribution are dealt with by the distribution itself.

By default, BCM aims to keep the distribution unchanged from its original release.

Security updates: Security updates are normally handled by package managers.

- Security updates that the distributions provide as package updates are handled by updating the distribution that underlies BCM.
- Security updates that BCM provides as package updates are handled by updating the BCM packages as usual.

In both cases the package managers (one of `yum`, `zypper`, or `apt`) are used to update the distribution (one of RHEL/Rocky, SUSE, or Ubuntu) that underlies BCM, or to update the BCM packages.

Security changes outside of BCM package management are normally outside the scope of BCM support.

Hardening: Hardening the distribution can include the following actions:

- removing packages
- blocking off ports
- disabling input hardware (USB ports, keyboards, mice...) in software
- adding disk encryption
- configuring SELinux
- adding binaries hardened in some way

Hardening is often possible but is not performed by default because BCM aims to keep the distribution as close to standard as possible.

Hardening can have unexpected side effects, including impacts on performance. Since hardening always involves balancing performance, usability, and security, it is the cluster administrator's responsibility to decide what to harden. Any hardening measures outside of BCM software typically fall outside the scope of BCM support.

An Aside About Upgrading The OS Or BCM

Upgrading from an earlier major release of the OS upon which BCM runs, to the next major release of the OS (for example, RHEL8 to RHEL9) while keeping BCM as it is, is not supported due to the OS dependencies that BCM has. If the OS is to be upgraded, then the recommendation is to install BCM once again from scratch on the upgraded OS.

Upgrading packages is only possible within the existing major OS release or existing BCM release.

Upgrading and updating are often ambiguously used as synonyms:

- When the OS upgrades to the next major release—for example, when the operating system is upgraded from RHEL8 to RHEL9—and the package is updated as part of that release, then updating the package is called a *package upgrade* or a *release upgrade*.
- Within an operating system major release—for example, when the operating system is still at a particular major version such as RHEL8 or RHEL9—updating a package is called a *package update*, or somewhat confusingly, also called a *package upgrade*, with the latter term being used in the context of it being the complement of a release upgrade.

Release upgrade packages are typically incompatible with packages from other releases, as many system administrators typically discover early on in their careers when they try to put a package update into a release upgrade, or the other way round. Even minor version package differences can be incompatible, as shown by the documentation and policies explaining compatibility levels (`https://access.redhat.com/articles/rhel8-abi-compatibility`, `https://access.redhat.com/articles/rhel9-abi-compatibility`). Letting the package manager sort it all out is what the sensible cluster administrator does.

Post-installation Software Management Typically Uses The Default Package Managers

Since BCM is built on top of an existing Linux distribution, the administrator should use the package utilities that are specific for the distribution (such as YUM and `rpm`, APT and `dpkg`, or YaST and Zypper) for software package management.

Packages managed by the distribution are hosted by distribution repositories. SUSE and RHEL distributions require the purchase of their license in order to access their repositories. The other distributions do not.

DGX OS, which is Ubuntu-based, relies on Ubuntu repositories. While BCM manuals cover much of DGX OS management, package and release upgrades for DGX OS are covered in the dedicated DGX OS documentation at

- `https://docs.nvidia.com/dgx/dgx-os-7-user-guide/upgrading-the-os.html` which covers upgrading an existing DGX OS 7.
- `https://docs.nvidia.com/dgx/dgx-os-7-user-guide/additional_software.html` which covers upgrading other software on DGX OS 7.

Packages managed by BCM are hosted by the BCM repository. Access to the BCM repositories also requires a license (Chapter 4 of the *Installation Manual*). Available packages for a particular BCM version and distribution can be viewed via the *package dashboard* at `https://service.bcm.nvidia.com/packages-dashboard/`.

Software Outside Of Default Package Management

There may also be software that the administrator would like to install that is outside the default packages collection. These could be source files that need compilation, or packages in other repositories.

Software Image Management

A software image (section 2.1.2) is a filesystem that a node picks up from a provisioner (a head node or a provisioning node) during provisioning so that the node can run as a linux system after provisioning. A subtopic of software management on a cluster is software image management—the management of software on a software image. By default, a node uses the same distribution as the head node for its base image along with necessary minimal, cluster-mandated changes. A node may however deviate from the default, and be customized by having software added to it in several ways.

Techniques Of Software Management Covered

This chapter covers the techniques of software management for the cluster.

Section 9.1 describes the naming convention for a BCM RPM or .deb package.

Section 9.2 describes how an RPM or .deb package is managed for the head node.

Section 9.3 describes how an RPM or .deb kernel package can be managed on a head node or image.

Section 9.4 describes how an RPM or .deb package can be managed on a software image.

Section 9.5 describes how a software other than an RPM or .deb package can be managed on a software image.

Section 9.6 describes how custom software images are created that are completely independent of the existing software image distribution and version.

Section 9.7 describes how multi-architecture and multi-distribution images can be created.

9.1 NVIDIA Base Command Manager Packages, Their Naming Convention And Version

Like the distributions it runs on top of, BCM uses

- either .rpm packages, managed by RPM (RPM Package Manager) or Zypper (ZYpp package manager)
- or .deb (Debian) packages, managed by APT (Advanced Package Tool)

For example, the `cmdaemon` package built by BCM has the following .rpm and .deb packages:

```
cmdaemon-HEAD-152061_cmHEAD_ec0ea0f4d1.x86_64.rpm    # for Rocky8 and SUSE
cmdaemon_HEAD-152061-cmHEAD-ec0ea0f4d1_amd64.deb     # for Ubuntu  BCM9.2
```

The file name has the following structure:

package-*version*-*revision*_cm*x*.*y*_*hash*.*architecture*.rpm

and

*package*_*version*-*revision*-cm*x*.*y*-*hash*_*architecture*.deb

where:

- *package* (`cmdaemon`) is the name of the package
- *version* (`HEAD`) is the version number of the package
- *revision* (`152061`) is the revision number of the package
- `cm` is used to indicate it is a package built by BCM for the cluster manager

- *x.y* (HEAD) is the version of BCM for which the RPM was built
- *hash* (ec0ea0f4d1) is a hash, and is only present for BCM packages. It is used for reference by the developers of BCM.
- *architecture* (x86_64 for RPMs or amd64 for APT) is the architecture for which the package was built. The architecture name of x86_64 or amd64 refers the same 64-bit x86 physical hardware in either case.

The differences in .rpm versus .deb package names are just some underbar/hyphen (_/-) changes, the hash (only for BCM packages), and the architecture naming convention.

Among the distributions supported by BCM, only Ubuntu uses .deb packages. The rest of the distributions use .rpm packages.

Querying The Packages

To check whether BCM or the distribution has provided a file that is already installed on the system, the package it has come from can be found.

For RPM-based systems: `rpm -qf` can be used with the full path of the file:

Example

```
[root@basecm11 ~]# rpm -qf /usr/bin/zless
gzip-1.9-9.el8.x86_64
[root@basecm11 ~]# rpm -qf /cm/local/apps/cmd/sbin/cmd
cmdaemon-HEAD-146965_cmHEAD_e6f593b676.x86_64
```

In the example, /usr/bin/zless is supplied by the distribution, while /cm/local/apps/cmd/sbin/cmd is supplied by BCM, as indicated by the "_cm" in the nomenclature.

For APT-based systems: A similar check can be done using `dpkg -S` to find the .deb package that provided the file, and then `dpkg -s` on the package name to reveal further information:

Example

```
[root@basecm11:~# dpkg -S /cm/local/apps/cmd/etc/cmd.env
cmdaemon: /cm/local/apps/cmd/etc/cmd.env
[root@basecm11:~# dpkg -s cmdaemon
Package: cmdaemon
Status: install ok installed
Priority: optional
Section: devel
Installed-Size: 78631
Maintainer: Cluster Manager Development <dev@brightcomputing.com>
Architecture: amd64
Version: HEAD-152061-cmHEAD-ec0ea0f4d1
Provides: cmdaemon
...
```

As an aside, system administrators should be aware that the BCM version of a package is provided and used instead of a distribution-provided version for various technical reasons. The most important one is that it is tested and supported by BCM. Replacing the BCM version with a distribution-provided version can result in subtle and hard-to-trace problems in the cluster, and support cannot be provided for a cluster that is in such a state, although some guidance may be given in special cases.

More information about the RPM Package Manager is available at http://www.rpm.org, while APT is documented for Ubuntu at http://manpages.ubuntu.com/manpages/.

9.1.1 The `packages` Command

BCM also provides the `packages` command in the `device` mode of `cmsh`. This should not be confused with the `packages` command used by `zypper`. The `packages` command used by BCM in `cmsh` queries devices directly for the installed packages independent of `rpm` or `deb` package management. It displays an overview of the package with its name name, type version, architecture, size, and installation date.

- The `-a|--all` option can be used to list all the packages installed on a particular node:

  Example

```
[basecm11]% device use node001
[basecm11->device[node001]]% packages -a
Node     Type    Name                Version             Arch     Size    Install date
-------- ------- ------------------ ------------------ -------- ------- --------------------
node001  deb     accountsservice    0.6.45-1ubuntu1    amd64    440kB   2019/02/14 10:51:06
node001  deb     acl                2.2.52-3build1     amd64    200kB   2019/02/14 10:51:19
node001  deb     acpid              1:2.0.28-1ubuntu1  amd64    139kB   2019/02/14 10:51:19
node001  deb     adduser            3.116ubuntu1       all      624kB   2019/02/14 10:49:53
...
```

- The `-c|--category` option can be used to list all the packages installed in a node category:

  Example

```
[basecm11]% device
[basecm11->device]% packages -a -c default
Node     Type   Name                Version             Arch     Size    Install date
-------- ------ ------------------ -------------------------- ------- --------------------
node001  deb    accountsservice    0.6.45-1ubuntu1    amd64    440kB   2019/02/14 10:51:06
node001  deb    acl                2.2.52-3build1     amd64    200kB   2019/02/14 10:51:19
...
node002  deb    accountsservice    0.6.45-1ubuntu1    amd64    440kB   2019/02/14 10:51:06
node002  deb    acl                2.2.52-3build1     amd64    200kB   2019/02/14 10:51:19
...
```

  Running the `-a` option for many nodes can be user-unfriendly. That is because per node this command typically returns about 100KB of data. So, for a 1000 nodes this would output about 100MB and a table with nearly a million lines.

  When checking packages for many nodes, it is best to request the package by name.

- Multiple `-f|--find` options can be used in the command line to display several packages.

  Example

```
[basecm11->device[node001]]% packages -f  cmdaemon -f cuda-dcgm-libs
Node     Type   Name            Version  Release               Arch     Size     Install date
-------- ------ --------------- -------- --------------------- -------- -------- --------------------
node001  deb    cmdaemon        1master  167241-cm-ea1a1840ef  amd64    89MiB    2026/01/21 13:27:50
node001  deb    cuda-dcgm-libs  1:4.4.2  100184-cm-179dca3bc0  amd64    346MiB   2026/01/21 13:27:49
```

- The `--outdated` option lists cluster manager packages that have upgrades available.

  Example

```
[basecm11->device]% packages --outdated
Node      Name                          Installed        Available
--------- ----------------------------- ---------------- ----------------
basecm11  cm-atftp-server               0.8.0-108        0.8.0-120
```

In the preceding, the installed `cm-atftp-server` package is older than the publicly available version.

- The `--outdated -a` option lists all the installed cluster manager packages and the available upgrades for the installed cluster manager packages.

  Example

```
[basecm11->device]% packages --outdated -a
Node      Name                          Installed        Available
--------- ----------------------------- ---------------- ----------------
basecm11  base-view                     11.0-111222      11.0-112166
basecm11  cluster-tools                 11.0-123245      11.0-123245
basecm11  cm-api-docs                   11.0-102138      11.0-102138
...
```

Further options include `-r|--rack`, `--role`, `--latest`, and `-t|--type`. Details and examples for these can be listed by running the `help packages` command within the `device` mode of `cmsh`.

9.1.2 BCM Package Point Release Versions And The `cm-package-release-info` Command

The `cm-package-release-info` command displays a precise package release version (package version) for each BCM package used in the cluster, and shows the BCM point release versions that should use that package version.

Background Information On BCM Version Nomenclature

Cluster manager version and release number: The *cluster manager version* is a *release number* with one decimal point—`10.0` in the preceding example. This is recorded in the file `/etc/cm-release`, and can also be seen in the output of the `versioninfo` command:

```
[root@basecm11 ~]# cmsh -c "main; versioninfo"
Version Information
------------------  ------
Cluster Manager     10.0
...
```

The release number version is the main way to refer to the software release series for a version. It is a tag that is associated with the whole collection of packages that is released as BCM. The release number 10.0 can be abbreviated to 10, and the release number for 11.0 can be abbreviated to 11. The release number should not be confused with the *point release number*, which is a number sequence with two decimal points.

Point release numbers: If a package manager is used then there is usually no need to know about point release numbers.

Point releases are releases that are part of the software release series for a version. Moving up to a new point release can have dependency issues. The package manager typically resolves these issues by keeping BCM packages correctly updated during a point release change. This means that typically a cluster administrator does not need to track the exact point release, and that referring to a point release during regular cluster administration is therefore typically avoided. Indeed, referring to a specific point release is often inappropriate, as discussed in `https://enterprise-support.nvidia.com/s/article/how-to-tell-what-bcm-version-are-you-running`, where it is pointed out that different packages may be updated to different point releases.

Point release number values: A point release number based on 10.0 is a numbering sequence with two decimal points, and might look like:

```
10.23.09
```

Point releases are interim releases, based on the main release number version, but with fixes and updates.

A point release based on the 10.0 and 11.0 releases can take two kinds of formats.

- Calendar-based versioning format:

  ```
  10.<YY>.<MM>  or
  11.<YY>.<MM>
  ```

  For example, for `10.23.09`, the `.0` from `10.0` is dropped for convenience, the year 2023 is indicated by `23`, and the month of September is indicated by `09`. Release numbers prior to 10.0 had other point release formats.

  Calendar-based versioning is used by

  - `10.0` release numbers up to and including release `10.25.03`
  - `11.0` release numbers up to and including release `11.25.08`

- Semantic versioning format:

  ```
  10.<MINOR>.<PATCH>  or
  11.<MINOR>.<PATCH>
  ```

  This follows the versioning method described in

  - `https://enterprise-support.nvidia.com/s/article/Base-Command-Manager-Semantic-Versioning#bcm-10` and
  - `http://semver.org`,

  and treats the prefixes 10 and 11 as major release numbers with incompatible APIs. The reason for switching from calendar versioning to semantic versioning is that it is hard to optimally synchronize the BCM release dates with the software release cycles of the other software that BCM has to work with. It meant that delays that were beyond the control of BCM development could happen, and result in release version numbers did not match with the calendar date.

  The reorganization to semantic versioning means that at the time of writing of this section (September 2025), the version numbers for point releases that are based on

  - the `10.0` release, and planned after `10.25.03` should be: `10.30.0`, `10.31.0`, `10.32.0`...
  - the `11.0` release, and planned after `11.25.08` should be: `11.30.0`, `11.31.0`, `11.32.0`...

  With semantic versioning for BCM, the number after the second point is typically 0 for a public release, and if that number is greater than 0 it should be a backward-compatible bugfix release.

When BCM has its next major change after the `11.0` release number—by major, the implication is that its APIs change—the BCM release number should then change to `12.0`, and the point release number should then roll over to `12.0.0`.

One more addition to the point release number is a letter suffix, in lower case, alphabetical order. This is typically for a "hotfix" release, where an existing point release is deemed to need an important fix right away, instead of having the fix wait until it goes into one of the next point release. For example,

if there is a 10.23.02 point release (itself a bugfix release), and it really needs a small change to its code to fix a significant problem a day after its initial release, then the release is given the label:

10.23.02a

In semantic versioning releases, letter suffixes are never used. In BCM they are ideally never used; however historically there have been hotfix releases.

Using The cm-package-release-info Command To Get A Point Release Version For A BCM Package

Yet, on some occasions it may be necessary to know the exact point release versions available for BCM packages. For example, in order to override dependencies if customizing a cluster in a non-standard way. Usually the point release information is needed for the main package, cmdaemon. Running the cm-package-release-info command displays the version of a BCM package, and for which point releases it is available for the current release number of the cluster.

Example

```
[root@basecm11 ~]# cm-package-release-info
Name                    Version   Release(s)
----------------------  --------  ------------------------------------------------------------------------
Lmod                    100094    10.24.03, 10.24.01, 10.23.09a, 10.23.10, 10.23.11, 10.23.12, 10.23.09
atftp-server            619       10.24.03
base-view               106987    10.24.03
blacs-openmpi-gcc-64    116       10.24.03, 10.24.01, 10.23.09a, 10.23.10, 10.23.11, 10.23.12, 10.23.09
blas-gcc-64             87        10.24.03, 10.24.01, 10.23.09a, 10.23.10, 10.23.11, 10.23.12, 10.23.09
bonnie++                83        10.24.03, 10.24.01, 10.23.09a, 10.23.10, 10.23.11, 10.23.12, 10.23.09
...
```

A package can be specified with the -f option:

Example

```
[root@basecm11 ~]# cm-package-release-info -f cmdaemon,cluster-tools
Name           Version  Release(s)
-------------  -------- ----------
cluster-tools  119838   10.23.11
cmdaemon       156713   10.23.10
```

9.2 Managing Packages On The Head Node

9.2.1 Managing RPM Or .deb Packages On The Head Node

Once BCM has been installed, distribution packages and BCM software packages are conveniently managed using the yum, zypper or apt repository and package managers. The zypper tool is recommended for use with the SUSE distribution, the apt utility is recommended for use with Ubuntu, and yum is recommended for use with the other distributions that BCM supports. YUM is not set up by default in SUSE, and installing and using it with SUSE is not recommended, unless the administrator is familiar with configuring YUM.

Listing Packages On The Head Node With YUM and Zypper

For YUM and zypper, the following commands list all available packages:

```
yum list
```
or
```
zypper refresh; zypper packages
```

For zypper, the short command option pa can also be used instead of packages.

Listing Packages On The Head Node With APT

For Ubuntu, the `apt-cache` command is used to view available packages. To generate the cache used by the command, the command:

```
apt-cache gencaches
```

can be run.

A verbose list of available packages can then be seen by running:

```
apt-cache dumpavail
```

It is usually more useful to use the `search` option to `apt-cache` to search for the package with a regex:

```
apt-cache search <regex>
```

A similar, but slightly more verbose option is the `search` option for `apt`:

```
apt search <regex>
```

Updating/Installing Packages On The Head Node

To install a new package called <*package name*> into a distribution, the corresponding package managers are used as follows:

```
yum install <package name>
zypper in <package name>        #for SLES
apt install <package name>  #for Ubuntu
```

Installed packages can be updated to the latest by the corresponding package manager as follows:

```
yum update
zypper refresh; zypper up      #refresh recommended to update package metadata
apt update; apt upgrade        #update recommended to update package metadata
```

An aside on the differences between the `update`, `refresh`/`up`, and `update`/`upgrade` options of the package managers: The `update` option in YUM by default installs any new packages. On the other hand, the `refresh` option in zypper, and the `update` option in APT only update the meta-data (the repository indices). Only if the meta-data is up-to-date will an `update` via zypper, or an `upgrade` via `apt` install any newly-known packages. For convenience, in the BCM manuals, the term update is used in the YUM sense in general—that is, to mean including the installation of new packages—unless otherwise stated.

Note for HA setups: In a high availability configuration, package updates that include `cmdaemon` must be applied to the active head node first, and only afterwards to the passive head node. This ordering, and the reason for it, is described in section 15.4.5.

The BCM repository has YUM and zypper repositories of its packages at:

```
http://updates.brightcomputing.com/yum
```

and updates are fetched by YUM and zypper for BCM packages from there by default, to overwrite older package versions by default.

For Ubuntu, the BCM .deb package repositories are at:

```
http://updates.brightcomputing.com/deb
```

Accessing the repositories manually (i.e. not using `yum`, `zypper`, or `apt`) requires a username and password. Authentication credentials can be provided upon request by opening a support ticket (section 14.2).

Cleaning Package Caches On The Head Node

The repository managers use caches to speed up their operations. Occasionally these caches may need flushing to clean up the index files associated with the repository. This can be done by the appropriate package manager with:

```
yum clean all
zypper clean -a   #for SUSE
apt-get clean     #for Ubuntu
```

Signed Package Verification

As an extra protection to prevent BCM installations from receiving malicious updates, all BCM packages are signed with the Bright Computing GPG public key (`0x5D849C16`), installed by default in `/etc/pki/rpm-gpg/RPM-GPG-KEY-cm` for Red Hat and derivaties. The Bright Computing public key is also listed in Appendix B.

The first time YUM or zypper are used to install updates, the user is asked whether the Bright Computing public key should be imported into the local repository packages database. Before answering with a "`Y`", `yum` users may choose to compare the contents of `/etc/pki/rpm-gpg/RPM-GPG-KEY-cm` with the key listed in Appendix B to verify its integrity. Alternatively, the key may be imported into the local RPM database directly, using the following command:

```
rpm --import /etc/pki/rpm-gpg/RPM-GPG-KEY-cm
```

With APT, the BCM keyring is already imported into `/etc/apt/trusted.gpg.d/brightcomputing-archive-cm.gpg` if the `cm-config-apt` package, provided by the Bright Computing repository, has been installed. The `cm-config-apt` package is installed by default for the Ubuntu edition of BCM.

Third Party Packages

The third party packages in the following list may be repackaged for BCM for installation purposes. The packages are described in Chapter 7 of the *Installation Manual*:

- Modules (section 7.1)
- Shorewall (section 7.2)
- GCC (section 7.3)

Exclusion of packages on the head node can be carried out as explained in section 9.3.2, where the kernel package is used as an example for exclusion.

9.2.2 Installation Of Packages On The Head Node That Are Not .deb And Not .rpm Packages

Sometimes a package is not packaged as an RPM or .deb package for BCM or for the distribution. In that case, the software can usually be treated as for installation onto a standard distribution. There may be special considerations on placement of components that the administrator may feel appropriate due to the particulars of a cluster configuration.

For example, for compilation and installation of the software, some consideration may be made of the options available on where to install parts of the software within the default shared filesystem. A software may have a compile option, say `--prefix`, that places an application <*application*> in a directory specified by the administrator. If the administrator decides that <*application*> should be placed in the shared directory, so that everyone can access it, the option could then be specified as: "`--prefix=/cm/shared/apps/`<*application*>".

Other commonly provided components of software for the applications that are placed in shared may be documentation, licenses, configuration settings, and examples. These may be placed in the directories `/cm/shared/docs`, `/cm/shared/licenses`, `/cm/shared/etc`, and `/cm/shared/examples`. The placement may be done with a compiler option, or, if that is not done or not possible, it could be done by modifying the placement by hand later. It is not obligatory to do the change of placement, but it helps with cluster administration to stay consistent as packages are added.

Module files (section 2.2 of this manual, and 7.1 of the *Installation Manual*) may sometimes be provided by the software, or created by the administrator to make the application work for users easily with the right components. The directory `/cm/shared/modulefiles` is recommended for module files to do with such software.

To summarize the above considerations on where to place software components, the directories under `/cm/shared` that can be used for these components are:

```
/cm/shared/
|-- apps
|-- docs
|-- etc
|-- examples
|-- licenses
`-- modulefiles
```

9.3 Kernel Management On A Head Node Or Image

Care should be taken when updating a head node or a software image. This is particularly true when custom kernel modules compiled against a particular kernel version are being used.

A package can be managed in a software image and the image deployed to nodes. A careful administrator typically clones a copy of a working image that is known to work, before modifying the image.

9.3.1 Installing A Standard Distribution Kernel Into An Image Or On A Head Node

A standard distribution kernel is treated almost like any other package in a distribution.

This means that:

- For head nodes, installing a standard kernel is done according to the normal procedures of managing a package on a head node (section 9.2).
- For regular nodes, installing a standard distribution kernel is done according to the normal procedures of managing an package inside an image, via a changed root (chroot) directory (section 9.4), but with some special aspects that are discussed in this section.

When a kernel is updated or reinstalled (section 9.3.3), kernel-specific drivers, such as OFED drivers may need to be updated or reinstalled. OFED driver installation details are given in Chapter 10 of the *Installation Manual*.

Kernel Package Name Formats

For RHEL, individual kernel package names take a form such as:

`kernel-3.10.0-327.3.1.el7.x86_64.rpm`

The actual one suited to a cluster varies according to the distribution used. RPM Packages with names that begin with "`kernel-devel-`" are development packages that can be used to compile custom kernels, and are not required when installing standard distribution kernels.

For Ubuntu, individual Linux kernel image package names take a form such as:

`linux-image-*.deb`

or

```
linux-signed-image-*.deb
```

Running `apt-cache search ^linux | grep 'kernel image'` shows the various packaged kernel images in the distribution.

Other Extra Considerations

When installing a kernel, besides the `chroot` steps of section 9.4, extra considerations for kernel packages are:

- The kernel must also be explicitly set in CMDaemon (section 9.3.3) before it may be used by the regular nodes.
- If using the `chroot` method to install the kernel rather than the `cm-chroot-sw-img` method (section 9.4.1), some other warnings to do with missing `/proc` paths may appear. For RHEL and derivatives, these warnings can be ignored.
- The ramdisk of a regular node must be regenerated using the `createramdisk` command (section 9.4.3).
- If the cluster is in a high availability configuration, then installing a new kernel on to the active head node may in some edge cases stop its network interface, and trigger a failover. It is therefore usually wiser to make the change on the passive head node first, or to disable automatic failover, before carrying out a change that could initiate a failover.

As is standard for Linux, both head or regular nodes must be rebooted to use the new kernel.

9.3.2 Excluding Kernels And Other Packages From Updates

Specifying A Kernel Or Other Package For Update Exclusion

Sometimes it may be desirable to exclude the kernel from updates on the head node.

- When using `yum`, to prevent an automatic update of a package, the package is listed after using the `--exclude` flag. So, to exclude the kernel from the list of packages that should be updated, the following command can be used:

  ```
  yum --exclude kernel update
  ```

  To exclude a package such as `kernel` permanently from all YUM updates, without having to specify it on the command line each time, the package can instead be excluded inside the repository configuration file. YUM repository configuration files are located in the `/etc/yum.repos.d` directory, and the packages to be excluded are specified with a space-separated format like this:

  `exclude = <package 1> <package 2> ...`

- The `zypper` command can also carry out the task of excluding the kernel package from getting updated when updating. To do this, the kernel package is first locked (prevented from change) using the `addlock` command, and the `update` command is run. Optionally, the kernel package is unlocked again using the `removelock` command:

  ```
  zypper addlock kernel
  zypper update
  zypper removelock kernel          #optional
  ```

- One APT way to upgrade the software while excluding the kernel image package is to first `update` the system, then to mark the kernel as a package that is to be held, and then to upgrade the system. Optionally, after the upgrade, the `hold` mark can be removed:

```
apt update
apt-mark hold <linux-image-version>
apt upgrade
apt-mark unhold <linux-image-version>      #optional
```

The complementary way to carry out an upgrade in APT while holding the kernel back, is to use *pinning*. Pinning can be used to set dependency priorities during upgrades. Once set, it can hold a particular package back while the rest of the system upgrades.

Specifying A Repository For Update Exclusion

Sometimes it is useful to exclude an entire repository from an update on the head node. For example, the administrator may wish to exclude updates to the parent distribution, and only want updates for the cluster manager for the OS to be pulled in. In that case, in RHEL-derivatives a construction such as the following may be used to specify that only the repository IDs matching the glob `cm*` are used, from the repositories in `/etc/yum.repos.d/`:

```
[root@basecm11 ~]# yum repolist
...
repo id                          repo name
appstream                        Rocky Linux 9 - AppStream
baseos                           Rocky Linux 9 - BaseOS
cm-rhel9-HEAD-updates            Cluster Manager HEAD for Red Hat Enterprise Linux 9 - Updates
crb                              Rocky Linux 9 - CRB
cuda-rhel9-x86_64                cuda-rhel9-x86_64
epel                             epel
epel-cisco-openh264              Extra Packages for Enterprise Linux 9 openh264 (From Cisco) - x86_64
extras                           Rocky Linux 9 - Extras
[root@basecm11 ~]# yum --disablerepo=* --enablerepo=cm* update
```

In Ubuntu, repositories can be added or removed by editing the repository sources under `/etc/apt/sources.list.d/`. There is also the `apt edit-sources` command, which, unsurprisingly, also edits the repository sources. The `add-apt-repository` command (`man add-apt-repository.1`) edits the repository sources by line. Running `add-apt-repository -h` shows options and examples.

9.3.3 Updating A Kernel In A Software Image

A kernel is typically updated in the software image by carrying out a package installation using the chroot environment (section 9.4), or specifying a relative root directory setting.

Package dependencies can sometimes prevent the package manager from carrying out the update, for example in the case of OFED packages (Chapter 10 of the *Installation Manual*). In such cases, the administrator can specify how the dependency should be resolved.

Parent distributions are by default configured, by the distribution itself, so that only up to 3 kernel images are kept when installing a new kernel with the package manager. However, in a BCM cluster, this default distribution value is overridden by a default BCM value, so that kernel images are never removed during YUM updates, or `apt upgrade`, by default.

For a software image, if the kernel is updated by the package manager, then the kernel is not used on reboot until it is explicitly enabled with either Base View or `cmsh`.

- To enable it using Base View, the `Kernel version` entry for the software image should be set. This can be accessed using the navigation path:

  `Provisioning > Software images` > `Settings > Kernel version`

  (figure 9.1).

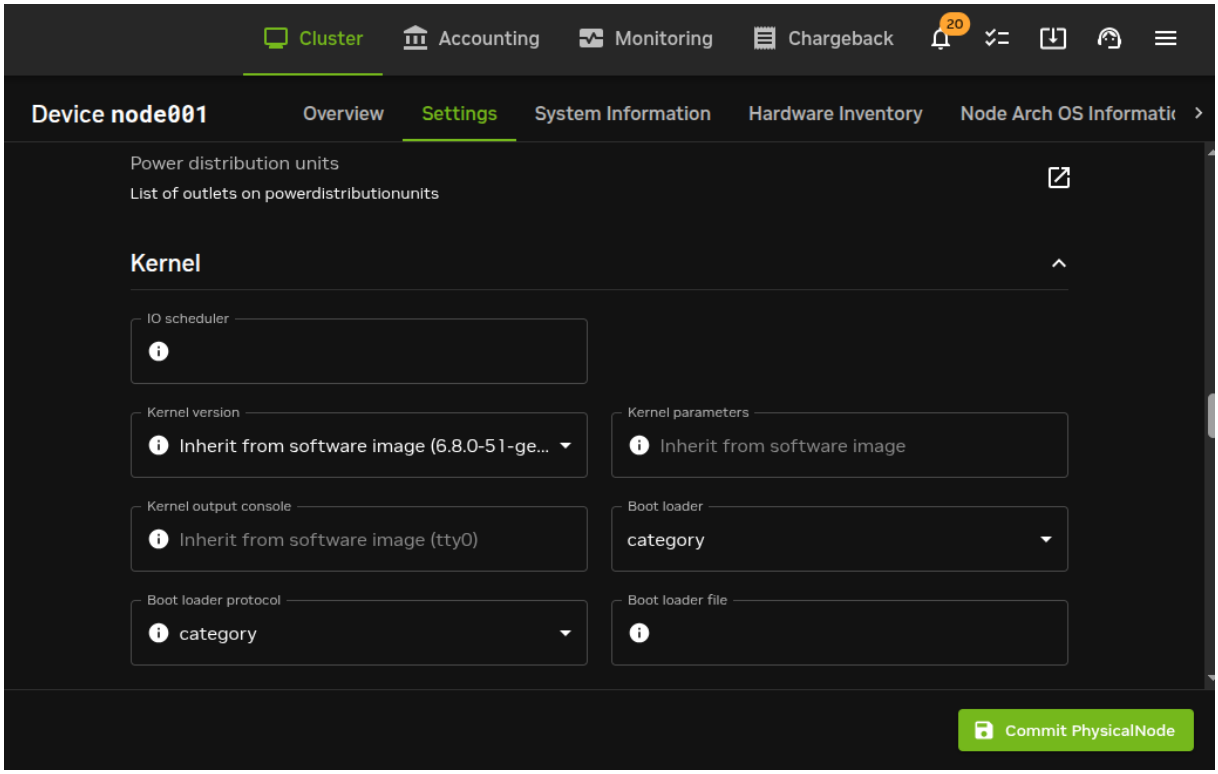


Figure 9.1: Updating A Software Image Kernel With Base View

- To enable the updated kernel from `cmsh`, the `softwareimage` mode is used. The `kernelversion` property of a specified software image is then `set` and committed:

Example

```
[root@basecm11 ~]# cmsh
[basecm11]% softwareimage
[basecm11]->softwareimage% use default-image
[basecm11->softwareimage[default-image]]% set kernelversion  3.10.0-327.3.1.el7.x86_64
[basecm11->softwareimage*[default-image*]]% commit -w
```

Tab-completion suggestions for the `set kernelversion` command will display the available values for the kernel version.

9.3.4 Setting Kernel Options For Software Images

A standard kernel can be booted with special options that alter its functionality. For example, a kernel can boot with `apm=off`, to disable Advanced Power Management, which is sometimes useful as a workaround for nodes with a buggy BIOS that may crash occasionally when it remains enabled.

In Linux in general, kernel parameters such as `apm=off` can be viewed in `/proc/cmdline` on a running system.

In Base View, to enable booting with this kernel option setting, the navigation path `Provisioning > Software images` > `Settings > Kernel parameters` (figure 9.1) is used to set the kernel parameter to `apm=off` for that particular image.

In `cmsh`, the equivalent method is to modify the value of "`kernel parameters`" in `softwareimage` mode for the selected image:

Example

```
[root@basecm11 ~]# cmsh
[basecm11]% softwareimage
[basecm11]->softwareimage% use default-image
[basecm11->softwareimage[default-image]]% append kernelparameters " apm=off"
[basecm11->softwareimage*[default-image*]]% commit
```

Often kernel options load up modules and their parameters. Making module loading persist after reboot and setting module loading order is covered in section 5.3.2

Some kernel options may require changes to be made in the BIOS settings in order to function.

9.3.5 Kernel Driver Modules

BCM provides some packages which install new kernel drivers or update kernel drivers. In RPM-based distributions, such packages generally require the `kernel-devel` package. In this section, the `kernel-devel-check` utility is first described, followed by the various drivers that BCM provides.

Kernel Driver Modules: `kernel-devel-check` Compilation Check

For RPM, the distribution's `kernel-devel` package is required to compile kernel drivers for its kernel. It must be the same version and release as the kernel running on the node. For APT, the `linux-header` package corresponding to the kernel image is used.

In RPM-based distributions, to check the head node and software images for the installation status of the `kernel-devel` package, the BCM utility `kernel-devel-check` is run from the head node:

Example

```
[root@mycluster ~]# kernel-devel-check
Head node: mycluster
  No kernel development directories found, probably no kernel development package installed.
package kernel-devel-3.10.0-957.1.3.el7.x86_64 is not installed
  Kernel development package kernel-devel-3.10.0-957.1.3.el7.x86_64 not found
  If needed, try to install the kernel development package with:
  # yum install kernel-devel-3.10.0-957.1.3.el7.x86_64

Software image: default-image
  No kernel development directories found, probably no kernel development package installed.
package kernel-devel-3.10.0-957.1.3.el7.x86_64 is not installed
  Kernel development package kernel-devel-3.10.0-957.1.3.el7.x86_64 not found
  If needed, try to install the kernel development package with:
  # chroot /cm/images/default-image yum install kernel-devel-3.10.0-957.1.3.el7.x86_64
```

As suggested by the output of `kernel-devel-check`, running a command on the head node such as:

```
[root@mycluster ~]# chroot /cm/images/default-image1 yum install \
kernel-devel-3.10.0-957.1.3.el7.x86_64
```

installs a `kernel-devel` package, to the software image called `default-image1` in this case. The package version suggested corresponds to the kernel version set for the image, rather than necessarily the latest one that the distribution provides.

Kernel Driver Modules: Improved Intel Wired Ethernet Drivers

Improved Intel wired Ethernet drivers—what they are: The standard RHEL and SLES distributions provide Intel wired Ethernet driver modules as part of the kernel they provide. BCM provides an improved version of the drivers with its own `intel-wired-ethernet-drivers` package. The package contains more recent versions of the Intel wired Ethernet kernel drivers: `e1000`, `e1000e`, `igb`, `igbvf`, `ixgbe` and `ixgbevf`. They often work better than standard distribution modules when it comes to performance, features, or stability.

Improved Intel wired Ethernet drivers—replacement mechanism: The improved drivers can be installed on all nodes.

For head nodes, the standard Intel wired Ethernet driver modules on the hard drive are overwritten by the improved versions during package installation. Backing up the standard driver modules before installation is recommended, because it may be that some particular hardware configurations are unable to cope with the changes, in which case reverting to the standard drivers may be needed.

For regular nodes, the standard distribution wired Ethernet drivers are not overwritten into the provisioner's software image during installation of the improved drivers package. Instead, the standard driver modules are removed from the kernel and the improved modules are loaded to the kernel during the `init` stage of boot.

For regular nodes in this "unwritten" state, removing the improved drivers package from the software image restores the state of the regular node, so that subsequent boots end up with a kernel running the standard distribution drivers from on the image once again. This is useful because it allows a very close-to-standard distribution to be maintained on the nodes, thus allowing better distribution support to be provided for the nodes.

If the software running on a fully-booted regular node is copied over to the software image, for example using the "`Grab to image`" menu item (section 9.5.2), this writes the improved driver module into the software image. Restoring to the standard version is then no longer possible with simply removing the improved drivers packages. This makes the image less close-to-standard, and distribution support is then less easily obtained for the node.

Thus, after the installation of the package is done on a head or regular node, for every boot from the next boot onward, the standard distribution Intel wired Ethernet drivers are replaced by the improved versions for fully-booted kernels. This replacement occurs before the network and network services start. The head node simply boots from its drive with the new drivers, while a regular node initially starts with the kernel using the driver on the software image, but then if the driver differs from the improved one, the driver is unloaded and the improved one is compiled and loaded.

Improved Intel wired Ethernet drivers—installation: The drivers are compiled on the fly on the regular nodes, so a check should first be done that the `kernel-devel` package is installed on the regular nodes (section 9.3.5).

If the regular nodes have the `kernel-devel` package installed, then the following `yum` commands are issued on the head node, to install the package on the head node and in the `default-image`:

Example

```
[root@mycluster ~]# yum install intel-wired-ethernet-drivers
[root@mycluster ~]# chroot /cm/images/default-image
[root@mycluster /]# yum install intel-wired-ethernet-drivers
```

For SUSE, the equivalent `zypper` commands are used ("`zypper in`" instead of "`yum install`").

Kernel Driver Modules: CUDA Driver Installation

CUDA drivers are drivers the kernel uses to manage GPUs. These are compiled on the fly for nodes with GPUs in BCM. The details of how this is done is covered in the CUDA software section (Chapter 9 of the *Installation Manual*).

Kernel Driver Modules: OFED Stack Installation

By default, the distribution provides the OFED stack used by the kernel to manage the InfiniBand or RDMA interconnect. Installing an NVIDIA DOCA OFED stack to replace the distribution version is covered in Chapter 10 of the *Installation Manual*. Some guidance on placement into initrd for the purpose of optional InfiniBand-based node provisioning is given in section 5.3.3.

9.4 Managing A Package In A Software Image And Running It On Nodes

A package can be managed in a software image and the image deployed to nodes. A careful administrator typically clones a copy of a working image that is known to work, before modifying the image.

9.4.1 Installing From Head Into The Image: Changing The Root Directory Into Which The Packages Are Deployed

Managing packages (including the kernel) inside a software image is most easily done while on the head node, using a "change root" (`chroot`) mechanism. The easiest way to carry out the `chroot` mechanism in BCM is to use a wrapper provided by BCM, `cm-chroot-sw-img`, which works with all distributions.

The same can be carried out more laboriously using the distribution package managers, such as `rpm`, `yum`, `zypper`, or `apt`, by using the associated `chroot` package manager option, or invoking `chroot` as a standalone command.

Change Root As An Option In The Package Manager Command

Using the `rpm` command: The `rpm` command supports the `--root` flag. To install an RPM package inside the default software image while in the head node environment, using the repositories of the head node, the command can be used as follows:

Example

```
rpm --root /cm/images/default-image -ivh /tmp/libxml2-2.6.16-6.x86_64.rpm
```

Using the `yum` command: The `yum` command allows more general updates with a change root option. For example, all packages in the default image can be updated using `yum` for RHEL and derivatives with:

Example

```
yum --installroot=/cm/images/default-image update    #for RHEL variants
```

A useful option to restrict the version to which an image is updated, is to use the option `--releasever`. For example, to allow only updates up to RHEL9.1, the command in the preceding example would have `--releasever=9.1` appended to it.

Using the `zypper` command: For SLES, `zypper` can be used as follows to update the image:

Example

```
zypper --root /cm/images/default-image up            #for SLES
```

Change Root With `chroot`, Then Running The Package Manager Commands

If the repositories used by the software image are the same as the repositories used by the head node, then the `chroot` command can be used instead of the `--installroot|--root` options to get the same result as the package manager options. That is, the same result is accomplished by first `chroot`ing into an image, and subsequently executing the `rpm`, `yum`, or `zypper` commands without `--root` or `--installroot` arguments. Thus:

For RHEL and derivatives: For YUM-based update, running `yum update` is recommended to update the image, after using the `chroot` command to reach the root of the image:

Example

```
[root@basecm11 ~]# chroot /cm/images/default-image
[root@basecm11 /]# yum update    #for RHEL variants
...updates happen...
[root@basecm11 /]# exit          #get out of chroot
```

For SLES: For SLES, running `zypper up` is recommended to update the image, after using the `chroot` command to reach the root of the image:

Example

```
basecm11:~# chroot /cm/images/default-image
basecm11.cm.cluster:/ # zypper up            #for SLES
...updates happen...
basecm11.cm.cluster:/ # exit                 #get out of chroot
```

For Ubuntu: For Ubuntu and APT, for package installation into the software image, there is often a need for the `/proc`, `/sys`, `/dev`, and perhaps other directories to be available within the `chroot` jail. Additionally, the `/proc` namespace used should not be that of the head node due to namespace issues that affect decision-making in some of the pre- and post-installation script bundled with the package.

Pre-configuring all this with bind mounting before going into the chrooted filesystem is a little tedious. Therefore the BCM utility, `cm-chroot-sw-img`, is strongly recommended to take care of this.

Thus, for Ubuntu, if the cluster administrator would like to run `apt update; apt upgrade` to update the image, then the recommended way to do it is to start the process with the `cm-chroot-sw-img` command:

Example

```
root@basecm11:~# cm-chroot-sw-img /cm/images/default-image
...messages indicate that the special directories have been mounted automatically, and a chroot jail has been entered...
root@basecm11:/# apt update; apt upgrade #for Ubuntu
...An upgrade session runs in the image root. Some administrator inputs may be needed...
root@basecm11:/# exit   #get out of chroot
...messages indicate that the special directories have been unmounted automatically...
```

The `cm-chroot-sw-img` wrapper is less needed in other distributions, with `yum` and `zypper` instead of `apt`. This is because the namespace issues are not so serious in with those other distributions. However even in those other distributions, it is cleaner to use the wrapper.

Excluding Packages And Repositories From The Image

Sometimes it may be desirable to exclude a package or a repository from an image.

- If using `yum --installroot`, then to prevent an automatic update of a package, the package is listed after using the `--exclude` flag. For example, to exclude the kernel from the list of packages that should be updated, the following command can be used:

  ```
  yum --installroot=/cm/images/default-image --exclude kernel update
  ```

  To exclude a package such as `kernel` permanently from all YUM updates, without having to specify it on the command line each time, the package can instead be excluded inside the repository configuration file of the image. YUM repository configuration files are located in the `/cm/images/default-image/etc/yum.repos.d` directory, and the packages to be excluded are specified with a space-separated format like this:

  `exclude = <package 1> <package 2> ...`

- The `zypper` command can also carry out the task of excluding a package from getting updated when during update. To do this, the package is first locked (prevented from change) using the `addlock` command, then the `update` command is run, and finally the package is unlocked again using the `removelock` command. For example, for the kernel package:

```
zypper --root /cm/images/default-image addlock kernel
zypper --root /cm/images/default-image update
zypper --root /cm/images/default-image removelock kernel
```

- For Ubuntu, the `apt-mark hold` command can be used to exclude a package. This is described in the particular case of excluding the kernel package earlier on, in section 9.3.2.
- Sometimes it is useful to exclude an entire repository from an update to the image. For example, the administrator may wish to exclude updates to the base distribution (the distribution packages used on the node, without the BCM packages), and only want BCM updates to be pulled into the image. In that case, a construction like the following may be used to specify that, for example, from the repositories listed in `/cm/images/default-image/etc/yum.repos.d/`, only the repositories matching the pattern `cm*` are used:

```
[root@basecm11 ~]# cd /cm/images/default-image/etc/yum.repos.d/
[root@basecm11 yum.repos.d]# yum --installroot=/cm/images/defaul\
t-image --disablerepo=* --enablerepo=cm* update
```

- For Ubuntu, excluding a repository can be carried out by removing the repository under `/etc/apt/sources.list.d/`. Slightly handier may be to use the `add-apt-repository` command, or the `apt edit-sources` command.

9.4.2 Installing From Head Into The Image: Updating The Node

If the images are in place, then the nodes that use those images cannot run those images until they have the changes placed on the nodes. Rebooting the nodes that use the software images is a straightforward way to have those nodes start up with the new images. Alternatively, the nodes can usually simply be updated without a reboot, using `imageupdate` (section 5.6), if no reboot is required by the underlying Linux distribution.

9.4.3 Installing From Head Into The Image: Possible Issues When Using `rpm --root`, `yum --installroot` Or `chroot`

- The update process on an image, when using YUM, zypper, or APT, will fail to start if the image is being provisioned by a provisioner at the time. The administrator can either wait for provisioning requests to finish, or can ensure no provisioning happens by locking the image (section 5.4.7), before running the update process. The image can then be updated. The administrator normally unlocks the image after the update, to allow image maintenance by the provisioners again.

  Example

```
[root@basecm11 ~]# cmsh -c "softwareimage lock default-image"
[root@basecm11 ~]# yum --installroot /cm/images/default-image update
[root@basecm11 ~]# cmsh -c "softwareimage unlock default-image"
```

- The `rpm --root` or `yum --installroot` command can fail if the versions between the head node and the version in the software image differ significantly. For example, installation from a RHEL8 head node to a RHEL9 software image is not possible with those commands, and can only be carried out with `chroot`.
- While installing software into a software image with an `rpm --root`, `yum --installroot` or with a `chroot` method is convenient, there can be issues if daemons start up in the image, or if the distribution installation scripts exit with errors due to being in an image environment rather than a real instance.

For example, installation scripts that stop and re-start a system service during a package installation may successfully start that service within the image's chroot jail and thereby cause related, unexpected changes in the image. Pre- and post- (un)install scriptlets that are part of RPM or APT packages may cause similar problems.

BCM's RPM and `.deb` packages are designed to install under chroot without issues. However packages from other repositories may cause the issues described. To deal with that, the cluster manager runs the `chrootprocess` health check, which alerts the administrator if there is a daemon process running in the image. The `chrootprocess` also checks and kills the process if it is a `crond` process.

- For some package updates, the distribution package management system attempts to modify the ramdisk image. This is true for kernel updates, many kernel module updates, and some other packages. Such a modification is designed to work on a normal machine. For a regular node on a cluster, which uses an extended ramdisk, the attempt does nothing.

  In such cases, a new ramdisk image must nonetheless be generated for the regular nodes, or the nodes will fail during the ramdisk loading stage during start-up (section 5.8.4).

  The ramdisk image for the regular nodes can be regenerated manually, using the `createramdisk` command (section 5.3.2).

- Trying to work out what is in the image from under chroot must be done with some care.

  For example, under chroot, running "`uname -a`" returns the kernel that is currently running—that is the kernel outside the chroot. This is typically not the same as the kernel that will load on the node from the filesystem under chroot. It is the kernel in the filesystem under chroot that an unwary administrator may wrongly expect to detect on running the `uname` command under chroot.

  To find the kernel version that is to load from the image, the software image kernel version property (section 9.3.3) can be inspected using the cluster manager with:

  Example

  ```
  cmsh -c "softwareimage; use default-image; get kernelversion"
  ```

9.4.4 Managing A Package In The Node-Installer Image

A special software image is the node-installer image. The node-installer image was introduced in NVIDIA Base Command Manager version 9.0, to make multiarch (section 9.7) possible.

The node-installer image is an image that, unsurprisingly, contains the node-installer (section 5.4). The default `/cm/node-installer` tree is a standalone image for that architecture. It requires updating just like the regular software image. So, for example, in YUM, the entire tree can be updated with:

```
chroot /cm/node-installer yum update
```

or

```
yum --installroot=/cm/node-installer update
```

while a particular package inside the image, such as `util-linux`, could be installed with:

```
yum --installroot=/cm/node-installer install util-linux
```

Updating the node-installer is recommended whenever there are updates available, in order to fix possible bugs that might affect the node-installer operations.

9.5 Managing Non-RPM Software In A Software Image And Running It On Nodes

Sometimes, packaged software is not available for a software image, but non-packaged software is. This section describes the installation of non-packaged software onto a software image in these two cases:

1. copying only the software over to the software image (section 9.5.1)
2. placing the software onto the node directly, configuring it until it is working as required, and syncing that back to the software image using BCM's special utilities (section 9.5.2)

In both cases, before making changes, a careful administrator typically clones a copy of a working image that is known to work, before modifying the image.

As a somewhat related aside, completely overhauling the software image, including changing the base files that distinguish the distribution and version of the image is also possible. How to manage that kind of extreme change is covered separately in section 9.6.

However, this current section (9.5) is about modifying the software image with non-RPM software while staying within the framework of an existing distribution and version.

In all cases of installing software to a software image, it is recommended that software components be placed under appropriate directories under `/cm/shared` (which is actually outside the software image).

So, just as in the case for installing software to the head node in section 9.2.2, appropriate software components go under:

```
/cm/shared/
|-- apps
|-- docs
|-- examples
|-- licenses
`-- modulefiles
```

9.5.1 Managing The Software Directly On An Image

The administrator may choose to manage the non-packaged software directly in the correct location on the image.

For example, the administrator may wish to install a particular software to all nodes. If the software has already been prepared elsewhere and is known to work on the nodes without problems, such as for example library dependency or path problems, then the required files can simply be copied directly into the right places on the software image.

The `chroot` command may also be used to install non-packaged software into a software image. This is analogous to the `chroot` technique for installing packages in section 9.4:

Example

```
cd /cm/images/default-image/usr/src
tar -xvzf /tmp/app-4.5.6.tar.gz
chroot /cm/images/default-image
cd /usr/src/app-4.5.6
./configure --prefix=/usr
make install
```

Whatever method is used to install the software, after it is placed in the software image, the change can be implemented on all running nodes by running the `updateprovisioners` (section 5.2.4) and `imageupdate` (section 5.6.2) commands.

9.5.2 Managing The Software Directly On A Node, Then Syncing Node-To-Image

Why Sync Node-To-Image?

Sometimes, typically if the software to be managed is more complex and needs more care and testing than might be the case in section 9.5.1, the administrator manages it directly on a node itself, and then makes an updated image from the node after it is configured, to the provisioner.

For example, the administrator may wish to install and test an application from a node first before placing it in the image. Many files may be altered during installation in order to make the node work with the application. Eventually, when the node is in a satisfactory state, and possibly after removing any temporary installation-related files on the node, a new image can be created, or an existing image updated.

Administrators should be aware that until the new image is saved, the node loses its alterations and reverts back to the old image on reboot.

The node-to-image sync can be seen as the converse of the image-to-node sync that is done using `imageupdate` (section 5.6.2).

The node-to-image sync discussed in this section is done using the `Grab to image` menu option from Base View, or using the "`grabimage`" command with appropriate options in `cmsh`. The sync automatically excludes network mounts and parallel filesystems such as Lustre and GPFS, but includes any regular disk mounted on the node itself.

Some words of advice and a warning are in order here

- The cleanest, and recommended way, to change an image is to change it directly in the node image, typically via changes within a chroot environment (section 9.5.1).
- Changing the deployed image running on the node can lead to unwanted changes that are not obvious. While many unwanted changes are excluded because of the `excludelistgrab*` lists during a node-to-image sync, there is a chance that some unwanted changes do get captured. These changes can lead to unwanted or even buggy behavior. The changes from the original deployed image should therefore be scrutinized with care before using the new image.
- For scrutiny, the `bash` command:

  ```
  vimdiff <(cd image1; find . | sort) <(cd image2; find . | sort)
  ```

  run from `/cm/images/` shows the changed files for image directories `image1` and `image2`, with uninteresting parts folded away. The `<(`*commands*`)` construction is called *process substitution*, for administrators unfamiliar with this somewhat obscure technique.

Node-To-Image Sync Using Base View

In Base View, the software on the running node can be saved to an image. To do this for a particular node, for example, `node001`, the `Grab to image` screen options can be navigated to using:

`Devices > All Devices > Physical Node[node001]`⋮ `>Software image > Grab to Image >` *options*

(figures 9.2 and 9.3):

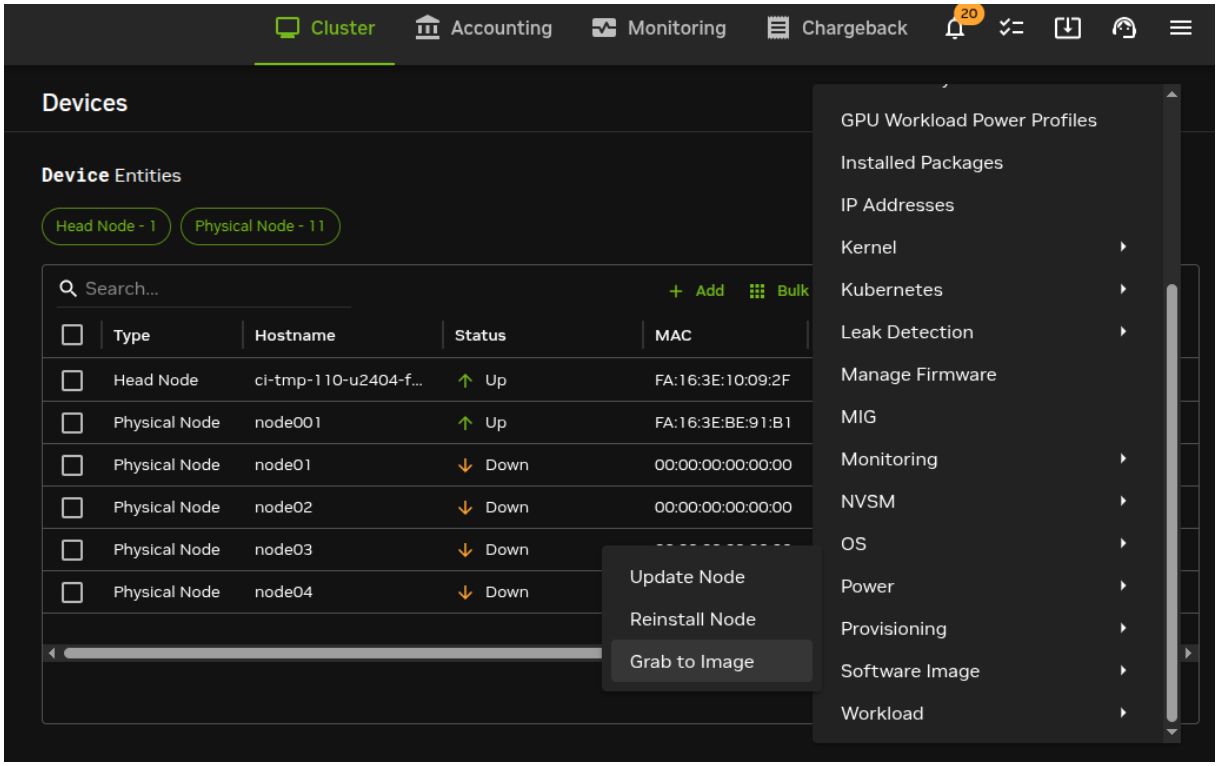


Figure 9.2: Synchronizing node-to-image: accessing `Grab to image`

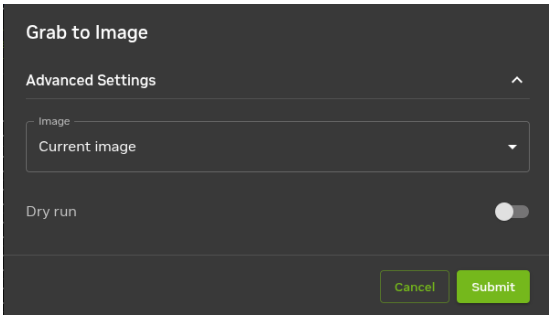


Figure 9.3: Synchronizing node-to-image: screen options for `Grab to image`

In the `Grab to image` screen (figure 9.3):

1. `Current image` can be selected. This is the image that the running node was provisioned from. Setting it synchronizes from the node back to the software image, using evaluation based on file change detection between the node and the image. It is thus a synchronization to the already existing software image that is currently in use by the node.

   The items that it excludes from the synchronization are specified in the `Exclude list image grab` text file, which is an exclude list originating in the category, and that can be viewed and edited using the navigation path:

   `Grouping > Categories` > `Settings > Exclude list grab`.

   This exclude list is known as `excludelistgrab` (page 543) in `cmsh`.

2. Another image can be selected. The image selected must already exist, and may be other than the image that the running node was provisioned from. Selection grabs what is to go to the image from the node. It wipes out whatever (if anything) is in the selected image, except for a list of excluded items.

   The excluded items can be viewed and edited in the `Exclude list grab new` block, navigated to using

   `Grouping > Categories` > `Settings > Exclude list grab new`.

   This exclude list is known as `excludelistgrabnew` (page 543) in `cmsh`.

The synchronization carried out when using the current image is a bit more "gentle" in carrying out the node-to-image sync, compared to the what is done when using the other image. That is, it carries out a "gentler sync" to avoid wiping out existing files, versus a "violent grab" to another image that can wipe out existing files. This means that there is a difference between using `Current image` and using `Other image` when both destination software images are the same.

The exclude lists are there to ensure, among other things, that the configuration differences between nodes are left alone for different nodes with the same image. The exclude lists are simple by default, but they conform in structure and patterns syntax in the same way that the exclude lists detailed in section 5.4.7 do, and can therefore be quite powerful.

If existing images are known to work well with nodes, then overwriting them with a new image on a production system may be reckless. A wise administrator who has prepared a node that is to write an image would therefore follow a process similar to the following instead of simply overwriting an existing image:

1. A new image can be cloned from the old image using the navigation path:

   `Provisioning > Software Images >` *old image* `> Clone`

   and setting a name for the new image, for example: `newimage`. The node state with the software installed on it would then be saved using the `Grab to image` option, and choosing the image name `newimage` as the image to save it to.

2. A new category is then cloned from the old category via the navigation path:

   `Grouping > Categories >` *the selected category*`> Clone`

   and setting a name for the new category, for example `newcategory`. The old image in `newcategory` is changed to the new image `newimage` using the navigation path:

   `Grouping > Categories > newcategory > Settings > Software image > newimage`

3. A newly-cloned category has no nodes initially. Some nodes are set to the new category so that their behavior with the new image can be tested. The chosen nodes can be made members of the new category from within the `Settings` option of each node, and saving the change. The navigation path for this is:

   `Devices > All Devices >` *the chosen nodes* `> Settings > Category > newcategory`

4. The nodes that have been placed in the new category are now made to pick up and run their new images. This can be done with a reboot of those nodes.

5. After sufficient testing, all the remaining nodes can be moved to using the new image. The old image is removed if no longer needed, or perhaps kept around just in case for reference.

Node-To-Image Sync Using cmsh

The preceding Base View method can alternatively be carried out using cmsh commands. The cmsh equivalent to the Grab to image with the Current image option is the grabimage command, available from device mode. The cmsh equivalent to the Grab to image with another image is the grabimage -i command, where the -i option specifies the image it will write to. As before, that image must be created or cloned beforehand.

The following cmsh session shows how a image is cloned, how a category is set for nodes that are to use the image, and how the running node with the new software on it is synchronized to the provisioning node that has the new image:

```
[root@basecm11 ~]# cmsh
[basecm11]% softwareimage
[basecm11->softwareimage]% clone default-image default-image1
[basecm11->softwareimage*[default-image1]]% commit
[basecm11->softwareimage[default-image1]]% category
[basecm11->category]% clone default default1
[basecm11->category*[default1*]]% commit
[basecm11->category[default1]]% set softwareimage default-image1
[basecm11->category*[default1*]]% commit
[basecm11->category[default1]]% device
[basecm11->device]% grabimage -w -i default-image1 node001
[basecm11->device]%
Mon Jul 18 16:13:00 2011 [notice] basecm11: Provisioning started on node node001
[basecm11->device]%
Mon Jul 18 16:13:04 2011 [notice] basecm11: Provisioning completed on node node001
```

The grabimage command without the -w option simply does a dry-run so that the user can see in the provisioning logs what should be grabbed, without having the changes actually carried out. Running grabimage -w instructs CMDaemon to really write the image.

When writing out the image, two exclude lists may be used:

- excludelistgrabnew: This is used with the grabimage commande run with the -i option. The list can be accessed and edited using the path:

  cmsh > category[*image*] > set excludelistgrabnew

  It corresponds to the Exclude list grab new exclusion list in Base View, on the navigation path:

  Devices > All Devices > Physical Node[node001] > Settings > Exclude list grab new

  which is associated with Grab to image (figure 9.3) in Base View.

- excludelistgrab: This is used with the grabimage command, run without the -i option. The list can be accessed and edited using the path:

  cmsh > category[*image*] > set excludelistgrab

  It corresponds to the Exclude list grab exclusion list in Base View, on the navigation path:

  Devices > All Devices > Physical Node[node001] > Settings > Exclude list grab

  which is associated with Grab to image (figure 9.3) in Base View.

9.6 Creating A Custom Software Image

By default, the software image used to boot non-head nodes is based on the same version and release of the Linux distribution as used by the head node. However, sometimes an image based on a different distribution or a different release from that on the head node may be needed.

A custom software image is created typically by building an entire filesystem image from a regular node. The node, which is never a head node, is then called the *base host*, with the term "base" used to indicate that it has no additional cluster manager packages installed. The distribution on the base host, is called the *base distribution* and is a selection of packages derived from the *parent distribution* (Red Hat, Scientific Linux etc.). A *base distribution package* is a package or rpm that is directly provided by the vendor of the parent distribution which the base distribution is based on, and is not provided by BCM.

Creating a custom software image consists of two steps. The first step (section 9.6.1) is to create a *base (distribution) archive* from an installed base host. The second step (section 9.6.2) is to create the image from the base archive using a special utility, `cm-create-image`.

An alternative to these steps is to use the `cm-image` tool (section 9.7.1), which is a wrapper to `cm-create-image`.

9.6.1 Creating A Base Distribution Archive From A Base Host

Structure Of The Base Distribution Archive

The step of creating the base distribution archive is done by creating an archive structure containing the files that are needed by the non-head node.

The filesystem that is archived in this way can differ from the special way that a Linux distribution unpacks and installs its filesystem on to a machine. This is because the distribution installer often carries out extra changes, for example in GRUB boot configuration. The creation of the base distribution archive is therefore a convenience to avoid working with the special logic of a distribution installer, which will vary across distributions and versions. Instead, the filesystem and contents of a node on which this parent distribution is installed—i.e. the end product of that logic—is what is dealt with.

The archive can be a convenient and standard `tar.gz` file archive (sometimes called the "base tar"), or, taking the step a little further towards the end result, the archive can be a fully expanded archive file tree. For convenience, a base tar is provided in the cluster installation ISO. For Ubuntu 24.04, the path on the ISO is `/data/UBUNTU2404.tar.gz`:

Example

```
joe@sandbox:~$ isoinfo -R -l -i bcm-11.0-ubuntu2404.iso | grep -A6 'Directory listing of /data/$'
Directory listing of /data/
drwxr-xr-x   1    0    0            2048 May  7 2025 [     22 02]  .
drwxr-xr-x   1    0    0            2048 May  7 2025 [     19 02]  ..
-rw-r--r--   1    0    0       370628239 May  7 2025 [ 198034 00]  bcminstallerfiles.tgz
-r--r--r--   1    0    0       147845120 May  7 2025 [ 379005 00]  filesystem.squashfs
drwxr-xr-x   1    0    0            2048 May  7 2025 [     23 02]  packages
-rw-r--r--   1    0    0      1182483439 May  7 2025 [5278054 00]  UBUNTU2404.tar.gz
joe@sandbox:~$
```

Repository Access Considerations When Intending To Build A Base Distribution Archive

For convenience, the archive should be up-to-date. So, the base host used to generate the base distribution archive should ideally have updated files. If, as is usual, the base host is a regular node, then it should ideally be up to date with the repositories that it uses. Therefore running `yum update` or `zypper up` on the base host image, and then provisioning the image to the base host, is recommended in order to allow the creation of an up-to-date base distribution archive.

However sometimes updates are not possible or desirable for the base host. This means that the base host archive that is put together from the base host filesystem is an un-updated archive. The custom image that is to be created from the archive must then be also be created without accessing the repositories, in order to avoid dependency problems with the package versions. Exclusion of access to the repositories is possible by specifying options to the `cm-create-image` command, and is described in section 9.6.2.

Examples Of How To Build A Base Distribution Archive

In the following example, a base distribution `tar.gz` archive `/tmp/BASEDIST.tar.gz` is created from the base host `basehost64`. The archive that is created should normally have access control lists and extended attributes preserved too:

Example

```
ssh root@basehost64 \
"tar -cz \
--exclude /etc/HOSTNAME --exclude /etc/localtime \
--exclude '/proc/*' --exclude /lost+found --exclude '/sys/*' \
--exclude /root/.ssh --exclude '/var/lib/dhcpcd/*' \
--exclude /media/floppy --exclude /etc/motd \
--exclude /root/.bash_history --exclude /root/CHANGES \
--exclude '/etc/udev/rules.d/*persistent*.rules' \
--exclude '/var/spool/mail/*' --exclude /rhn \
--exclude /etc/sysconfig/rhn/systemid --exclude '/tmp/*' \
--exclude '/var/spool/up2date/*' --exclude '/var/log/*' \
--exclude /etc/sysconfig/rhn/systemid.save \
--exclude /root/mbox --exclude '/var/cache/yum/*' \
--exclude /etc/cron.daily/rhn-updates /" > /tmp/BASEDIST.tar.gz
```

Or alternatively, a fully expanded archive file tree can be created from `basehost64` by `rsyncing` to an existing directory (here it is `/cm/images/new-image`):

Example

```
rsync -av  --hard-links --numeric-ids \
--exclude=/etc/HOSTNAME --exclude=/etc/localtime --exclude='/proc/*' \
--exclude=/lost+found --exclude='/sys/*' --exclude=/root/.ssh \
--exclude='/var/lib/dhcpcd/*' --exclude=/media/floppy \
--exclude=/etc/motd --exclude=/root/.bash_history \
--exclude=/root/CHANGES --exclude='/var/spool/mail/*' \
--exclude='/etc/udev/rules.d/*persistent*.rules' \
--exclude=/rhn --exclude=/etc/sysconfig/rhn/systemid \
--exclude=/etc/sysconfig/rhn/systemid.save --exclude='/tmp/*' \
--exclude='/var/spool/up2date/*' --exclude='/var/log/*' \
--exclude=/root/mbox --exclude='/var/cache/yum/*' \
--exclude=/etc/cron.daily/rhn-updates \
root@basehost64:/ /cm/images/new-image/
```

SELinux and file attributes: To use SELinux on compute nodes, extended attributes must not be used.

The defaults can be modified, if needed, by adjusting attributes for partitions via `cmsh`, in `fspart` mode:

Example

```
[basecm11->fspart]% foreach * (set rsyncxattr no)
[basecm11->fspart*]% list -f path:0,rsyncxattr
path (key)                   rsyncxattr
---------------------------- --------------------
/cm/images/default-image     no
/cm/images/default-image/boot no
/cm/node-installer           no
/cm/shared                   no
/tftpboot                    no
/var/spool/cmd/monitoring    no
```

Having built the archive by following the examples suggested, the first step in creating the software image is now complete.

9.6.2 Creating The Software Image With `cm-create-image`

The second step, that of creating the image from the base archive, now needs to be done. This uses the `cm-create-image` utility, which is part of the `cluster-tools` package.

The `cm-create-image` utility uses the base archive as the base for creating the image. By default, it expects that the base distribution repositories be accessible just in case files need to be fetched from a repository package.

Thus, when the `cm-create-image` utility is run with no options, the image created mostly picks up the software only from the base archive. However, the image picks up software from the repository packages:

- if it is required as part of a dependency, or
- if it is specified as part of the package selection file (page 549).

If a repository package file is used, then it should be noted that the repository package files may be more recent compared with the files in the base archive. This can result in an image with files that are perhaps unexpectedly more recent in version than what might be expected from the base archive, which may cause compatibility issues. To prevent this situation, the `--exclude` option (section 9.2) can be used to exclude updates for the packages that are not to be updated.

Repository access can be directly to the online repositories provided by the distribution, or it can be to a local copy. For RHEL, online repository access can be activated by registering with the Red Hat Network (section 5.1 of the *Installation Manual*). Similarly, for SUSE, online repository access can be activated by registering with SUSE (section 5.2 of the *Installation Manual*). An offline repository can be constructed as described in section 9.6.3 of this manual.

Usage Of The `cm-create-image` Command

The usage information for `cm-create-image` is:

```
[root@head ~]# cm-create-image -h
usage: cm-create-image
         [-h] [-a <archive>] [-d <dir>] [--fromhost <hostname>] [--frombfb <file>] [-k] [--cmdvd <path>]
         [--add-only] [--arch <arch>] [--os <os>] [-c <file>] [-b <file>] [--dgx] [--dgx-repo <file>]
         [--dgx-type {dgx_h100,dgx_b200,dgx_a100,dgx_gb200,dgx_h200,dgx_b300,dgx_gb300}] [-e] [-f]
         [-g <value>] [-i <dir>] [-j <list>] [--holdpackages <list>] [--no-holdpackages] [-l <file>]
         [-m] [-n <name>] [-o <file>] [-q <list>] [-r] [-s] [-t {node-installer,cmshared}] [-u] [-v]
         [-w <vendor>] [-x <list>] [-y] [-z <file>] [--no-progress] [-L <file>]
         [--sles-allow-vendor-change] [--skip-connectivity-check] [--keep-online-repos]
         [--tar-options ...] [--no-cm-repo-extra] [--cmshared-reinstall] [--no-cm-cuda-repo]
         [--rhsm-username <username>] [--rhsm-password <password>] [--scc-email <email>]
         [--scc-code <code>] [--extra-pkg-group <pkg-group>] [--force-chroot]
         [positional ...]
```

The options in the preceding usage are described further in the help text.

Examples In Usage Of `cm-create-image`

Explanations of the usage text follow:

1. In the following, a base distribution archive file, `/tmp/ROCKY9.tar.gz`, is written out to a software image named `rocky9-image`:

```
cm-create-image --fromarchive /tmp/ROCKY9.tar.gz --imagename rocky9-image
```

The image with the name rocky9-image is created in the CMDaemon database, making it available for use by cmsh and Base View. If an image with the name already exists, then cm-create-image exits and advises the administrator to provide an alternate name.

By default, the image name specified sets the directory into which the software image is installed. Thus here the directory is /cm/images/rocky9-image/.

2. Instead of the image getting written into the default directory as in the previous item, an alternative directory can be specified with the --imagedir option. Thus, in the following, the base distribution archive file, /tmp/ROCKY9.tar.gz is written out to the /cm/images/test-image directory. The software image is given the name rocky9-image:

```
cm-create-image --fromarchive /tmp/ROCKY9.tar.gz --imagename rocky9-image --imagedir \
/cm/images/test-image
```

3. If the contents of the base distribution file tree have been transferred to a directory, then no extraction is needed. The --fromdir option can then be used with that directory. Thus, in the following, the archive has already been transferred to the directory /cm/images/SLES15-image, and it is that directory which is then used to place the image under a directory named /cm/images/sles15-image/. Also, the software image is given the name sles15-image:

```
cm-create-image --fromdir /cm/images/SLES15-image --imagename sles15-image
```

   - **skipping:** Sometimes the installation of additional base distribution packages may need to be skipped.

     For example, if the target image already has the required base distribution packages, as in DGX OS, which already includes Ubuntu. In that case the --skipdist option must be used to skip package installation:

     ```
     cm-create-image --fromdir /cm/images/dgx-image --imagename dgx-image --skipdist
     ```

   - **updating:** If the software image already exists in CMDaemon, and if the target directory contents need to be updated, then the --updateimage option can be used:

     ```
     cm-create-image --fromdir /cm/images/dgx-image --imagename dgx-image --updateimage
     ```

   - **skipping and updating**: Applying both options means skipping the installation of additional base distribution packages, and then updating the software image, which can be an efficient way to set up an up-to-date DGX image:

     ```
     cm-create-image --fromdir /cm/images/login-image-a100-test --imagename \
     login-image-a100-test --updateimage --skipdist
     ```

4. A software image can be created from a running node using the --fromhost option. This option makes cm-create-image behave in a similar manner to grabimage (section 9.5.2) in cmsh. It requires passwordless access to the node in order to work. Generic nodes, that is nodes that are not managed by BCM, can also be used. An image named node001-image can then be created from a running node named node001 as follows:

```
cm-create-image --fromhost node001 --imagename node001-image
```

   By default the image goes under the /cm/images/node001-image/ directory.

5. The --basedistrepo flag is used together with a .repo file. The file defines the base distribution repository for the image. The file is copied over into the repository directory of the image, (/etc/yum.repos.d/ for Red Hat and similar, or /etc/zypp/repos.d/ for SLES).

6. The `--cmrepo` flag is used together with a `.repo` file. The file defines the cluster manager repository for the image. The file is copied over into the repository directory of the image, (`/etc/yum.repos.d/` for Red Hat and similar, or `/etc/zypp/repos.d/` for SLES).

7. A default node software image can be created with:

```
cm-create-image --imagename default-image --fromarchive <path to base archive> ...
```

8. A default node-installer image can be created with:

```
cm-create-image --imagename node-installer --image-type node-installer --fromarchive\
<path to base archive> ...
```

9. A default DGX platform image can be created from a vanilla Ubuntu 24.04 basetar with:

```
cm-create-image --fromarchive=/mnt/install/UBUNTU2404.tar.gz --imagedir=/cm/images/dgxos-image \
--imagename=ubuntu2404 --dgx
```

10. The `--extra-pkg-group` option lets some extra packages be specified for installation into an Ubuntu image.

    At the time of writing of this section (August 2025), the packages are DOCA OFED packages, as returned by the following `grep` command on the head node:

```
grep -E -o  "doca_ofed_[0-9.-]*" /cm/local/apps/cluster-tools/config/UBUNTU2404-dist-extrapackages.xml
doca_ofed_2.9.1-2
doca_ofed_2.9.1-3.0.2
doca_ofed_2.9.3
doca_ofed_2.10.0-093520
doca_ofed_3.0.0
```

    The XML file being grepped is a package selection file containing a list of extra packages. Package selection files (page 549) used for package selection with `cm-create-image` are XML files.

    Another way to install DOCA OFED is using the `cm-doca-ofed-install.sh` script-based installation method (section 10.2 of the *Installation Manual*).

    If there is already a DOCA OFED package in the image, then the `cm-doca-ofed-install.sh` method reports the version that is already there. Only if the `--force` option is used, does installation continue.

11. The `--dgx-type` option can be set to one of the following:

    - `dgx_h100` for the H100 GPU
    - `dgx_b200` for the B200 GPU
    - `dgx_a100` for the A100 GPU
    - `dgx_gb200` for the GB200 GPU
    - `dgx_h200` for the H200 GPU
    - `dgx_gb300` for the GB300 GPU
    - `dgx_b300` for the B300 GPU

    This sets the correct kernel parameters for the DGX image in a standardized manner.

Package Selection Files In `cm-create-image`

In the preceding explanations text, the selection of packages on the head node is done using a *package selection file*.

Package selection files are available in `/cm/local/apps/cluster-tools/config/`. For example, if the base distribution of the software image being created is Rocky Linux 8, then the configuration file used is:

```
/cm/local/apps/cluster-tools/config/ROCKY8-config-dist.xml
```

The package selection file is made up of a list of XML elements, specifying the image type of the package, its name, and architecture. For example:

```
...
<package image="master" name="adwaita-cursor-theme" arch="noarch" platforms="x86_64 aarch64" />
<package image="master" name="adwaita-gtk2-theme" arch="platform" platforms="x86_64" />
<package image="master" name="adwaita-icon-theme" arch="noarch" platforms="x86_64 aarch64" />
<package image="master" name="alsa-lib" arch="platform" platforms="x86_64 aarch64" />
...
```

The minimal set of packages in the list defines the minimal distribution that works with BCM, and is the base-distribution set of packages, which may not work with some features of the distribution or BCM. To this minimal set the following packages may be added to create the custom image:

- Packages from the standard repository of the parent distribution. These can be added to enhance the custom image or to resolve a dependency of BCM. For example, in the (parent) Red Hat distribution, packages can be added from RHEL's BaseOS and Appstream repositories to the base-distribution.
- Packages from outside the standard repository, but still from inside the parent distribution. These can be added to enhance the custom image or to resolve a dependency of BCM. For example, within the RHEL8 and RHEL9 distribution, unsupported packages can be added from the CRB (CodeReady Linux Builder) and Supplementary repositories to the base-distribution to enhance the capabilities of the image or resolve dependencies of BCM. Section 9.1 of the *Installation Manual* considers an example of such a dependency for the CUDA package.

Unless the required distribution packages and dependencies are installed and configured, particular features of BCM, such as CUDA, cannot work correctly or cannot work at all.

The package selection file also contains entries for the packages that can be installed on the head (`image="master"`) node. Therefore non-head node packages must have the `image="slave"` attribute.

Kernel Module Selection By `cm-create-image`

For an image created by `cm-create-image`, with a distribution <*dist*>, the default list of kernel modules to be loaded during boot are read from the file `/cm/local/apps/cluster-tools/config/`<*dist*>`-slavekernelmodules`.

<*dist*> can take the value `RHEL8U10`, `RHEL9U6`, `RHEL9U7`, `RHEL9U8`, `ROCKY8U10`, `ROCKY9U6`, `ROCKY9U7`, `ROCKY9U8`, `SLES15`, `SLES15SP6`, `SLES15SP7`, `UBUNTU2204`, `UBUNTU2404`, `UBUNTU2604`.

If custom kernel modules are to be added to the image, they can be added to this file.

Output And Logging During A `cm-create-image` Run

The `cm-create-image` run goes through several stages: validation, sanity checks, finalizing the base distribution, copying the BCM repository files, installing distribution packages, finalizing image services, and installing the BCM packages. An indication is given if any of these stages fail.

Further detail is available in the logs of the `cm-create-image` run, which are kept in logs of the form `/var/log/cm-create-image-`<*image name*>.log, where <*image name*> is the name of the built image.

Default Image Location

The default-image is at `/cm/images/default-image`, so the image directory can simply be kept as `/cm/images/`.

During a `cm-create-image` run, the `--imagedir` option allows an image directory for the image to be specified. This must exist before the option is used.

More generally, the full path for each image can be set:

- With Base View, using the navigation path `Provisioning > Software Images >` *a chosen image* `> Settings > Path`
- With `cmsh` within `softwareimage` mode, for example:

```
[basecm11->softwareimage]% set new-image path /cm/higgs/new-images
```

- At the system level, the images or image directory can be symlinked to other locations for organizational convenience

Removal Of `doca.list` Repositories List For DGX Images Created By `cm-create-image` Versions Earlier Than BCM 11.30.0

For versions of BCM prior to 11.30.0, for B200, GB200 and similar hardware, the repositories list for DOCA should be removed from the DGX image. If the file is kept in, then `cm-kubernetes-setup` (Chapter 4.2 of the *Containerization Manual*) fails at around progress precentage 22.

To remove the list of DOCA repositories, the following procedure can be carried out from the head node for a DGX image `dgx-image`:

```
cm-chroot-sw-img /cm/images/dgx-image
...
rm /etc/apt/sources.list.d/doca.list
ctrl-d to exit from image
reboot nodes to get updated image
```

The setup for Kubernetes can then be (re-)run.

9.6.3 Configuring Local Repositories For Linux Distributions, And For The BCM Package Repository, For A Software Image

Using local instead of remote repositories can be useful in the following cases:

- for clusters that have restricted or no internet access.
- for the RHEL and SUSE Linux distributions, which are based on a subscription and support model, and therefore do not have free access to their repositories.
- for creating a custom image with the `cm-create-image` command introduced in section 9.6.2, using local base distribution repositories.

The administrator can choose to access an online repository provided by the distribution itself via a subscription as described in Chapter 5 of the *Installation Manual*. Another way to set up a repository is to set it up as a local repository, which may be offline, or perhaps set up as a locally-controlled proxy with occasional, restricted, updates from the distribution repository.

In the four procedures that follow, the first two procedures explain how to create and configure a local offline SLES zypper or RHEL YUM repository for the subscription-based base distribution packages. These first two procedures assume that the corresponding ISO/DVD has been purchased/downloaded from the appropriate vendors. The third procedure then explains how to create a local offline YUM repository from the BCM ISO for CentOS so that a cluster that is completely offline still has a complete and consistent repository access. The fourth procedure explains the equivalent APT-based approach for Ubuntu.

Thus, a summary list of what these procedures are about is:

- Setting up a local repository for SLES (page 551)
- Setting up a local repository for RHEL (page 551)
- Setting up a local repository for CentOS and BCM from the BCM ISO for CentOS (page 551)
- Setting up a local repository for Ubuntu and BCM from the BCM ISO for Ubuntu (page 552)

Configuring Local Repositories For SLES For A Software Image

For SLES15 the required packages are available in two ISOs. Assuming that the image directory is `/cm/images/sles15sp5-image`, while the names of the ISOs are `SLE-15-SP5-Full-x86_64-GM-Media1.iso` and `SLE-15-SP5-Full-x86_64-GM-Media2.iso`, then the contents of the ISOs can be copied as follows:

```
mkdir /mnt1 /mnt2
mkdir /cm/images/sles15sp5-image/root/repo1
mkdir /cm/images/sles15sp5-image/root/repo2
mount -o loop,ro SLE-15-SP5-Full-x86_64-GM-Media1.iso /mnt1
cp -ar /mnt1/* /cm/images/sles15sp5-image/root/repo1/
mount -o loop,ro SLE-15-SP5-Full-x86_64-GM-Media2.iso /mnt2
cp -ar /mnt2/* /cm/images/sles15sp5-image/root/repo2/
```

The two repositories can be added for use by `zypper` in the image, as follows:

```
chroot /cm/images/sles15sp5-image
zypper addrepo /root/repo1 "SLES15SP5-MEDIA1"
zypper addrepo /root/repo2 "SLES15SP5-MEDIA2"
exit (chroot)
```

Configuring Local Repositories For RHEL For A Software Image

For RHEL distributions, the procedure is almost the same. The required packages are contained in one DVD.

```
mkdir /mnt1
mkdir /cm/images/rhel-image/root/repo1
mount -o loop,ro RHEL-DVD1.iso /mnt1
cp -ar /mnt1/* /cm/images/rhel-image/root/repo1/
```

The repository is added to YUM in the image, by creating the repository file `/cm/images/rhel-image/etc/yum.repos.d/rhel-base.repo` with the following contents:

```
[base]
name=Red Hat Enterprise Linux $releasever - $basearch - Base
baseurl=file:///root/repo1/Server
gpgcheck=0
enabled=1
```

Configuring Local Repositories For CentOS And BCM For A Software Image

Mounting the ISOs The variable `$imagedir` is assigned as a shortcut for the software image that is to be configured to use a local repository:

```
imagedir=/cm/images/default-image
```

If the ISO is called `basecom-centos.iso`, then its filesystem can be mounted by the root user on a new mount, `/mnt1`, as follows:

```
mkdir /mnt1
mount -o loop basecom-centos.iso /mnt1
```

The head node can then access the ISO filesystem.

The same mounted filesystem can also be mounted with the `bind` option into the software image. This can be done (from outside the chroot jail) inside the software image by the root user, in the same relative position as for the head node, as follows:

```
mkdir $imagedir/mnt1
mount -o bind /mnt1 $imagedir/mnt1
```

This allows an operation run under the `$imagedir` in a chroot environment to access the ISO filesystem too.

Creating YUM repository configuration files: YUM repository configuration files can be created:

- **for the head node:** A repository configuration file

  `/etc/yum.repos.d/cmHEAD-dvd.repo`

  can be created, for example, for a release tagged with a <*subminor*> number tag, with the content:

  ```
  [BCM-repo]
  name=NVIDIA Base Command Manager DVD Repo
  baseurl=file:///mnt1/data/packages/HEAD-<subminor>
  enabled=1
  gpgcheck=1
  exclude = slurm* cm-hwloc
  ```

- **for the regular node image:** A repository configuration file

  `$imagedir/etc/yum.repos.d/cmHEAD-dvd.repo`

  can be created. This file is in the image directory, but it has the same content as the previous head node yum repository configuration file.

Verifying that the repository files are set up right: To verify the repositories are usable on the head node, the YUM cache can be cleaned, and the available repositories listed:

```
[root@basecm11 ~]# yum clean all
[root@basecm11 ~]# yum repolist -v
BCM-repo                 NVIDIA Base Command Manager DVD Repo
...
```

To carry out the same verification on the image, these commands can be run with `yum --installroot=$imagedir` substituted in place of just `yum`.

The ISO repository should show up, along with any others that are accessible. Connection attempts that fail to reach a network-based or local repositories display errors. If those repositories are not needed, they can be disabled from within their configuration files.

Configuring Local Repositories For Ubuntu For A Software Image

Mounting the ISO: If the BCM Ubuntu ISO is called `bcm-ubuntu.iso`, then its filesystem can be mounted by the root user on a new mount, `/mnt1`, as follows:

```
mkdir /mnt1
mount -o loop,ro bcm-ubuntu.iso /mnt1
```

Creating an APT repository list file: Unlike YUM and zypper, which use `.repo` files under `/etc/yum.repos.d/`, APT repository sources are configured in `.list` files under `/etc/apt/sources.list.d/`. A list file can be created for the head node, for example `/etc/apt/sources.list.d/cm-dvd.list`, with content along the lines of:

```
deb file:///mnt1/data/packages/HEAD-<subminor>/ubuntu/<ubuntu version>/ ./
deb [trusted=yes] file:///mnt1/data/packages/dist/ubuntu/<ubuntu version>/ ./
```

The first line makes the BCM packages for that release available. As covered in section 9.6.3, the `cm-config-apt` package, which is installed by default on the Ubuntu edition of BCM, already imports the BCM keyring into `/etc/apt/trusted.gpg.d/brightcomputing-archive-cm.gpg`, so this line does not need `trusted=yes` to pass APT's signature check.

The second line, for the base Ubuntu distribution packages carried on the same ISO, is optional if the cluster has online access to the standard Ubuntu repositories, but is recommended for an offline or air-gapped cluster so that dependencies required by BCM packages, such as `cmdaemon`, can also be resolved locally. Unlike the BCM packages on the first line, these base distribution packages are not covered by the BCM keyring, so `trusted=yes` is used here to accept them without a signature check. This is a deliberate risk acceptance for local, physically-controlled installation media, not a default that should be copied to a network-hosted repository.

Applying the repository to a software image: To use the new repository inside a software image rather than just on the head node, the list file is copied into the image, and the ISO mount is bind-mounted into the image so that the image can reach the same files:

```
cp /etc/apt/sources.list.d/cm-dvd.list $imagedir/etc/apt/sources.list.d/
mkdir $imagedir/mnt1
mount -o bind /mnt1 $imagedir/mnt1
```

where `$imagedir` is the software image directory, as used in the preceding CentOS example. Both the `/mnt1` mount and its bind mount into `$imagedir/mnt1` are not persistent across a reboot. If the head node is rebooted, both mounts must be recreated before `apt update` is run again, in either the head node or the image, or the `file:///mnt1/...` paths in the `.list` file will fail to resolve.

Verifying that the repository is set up right: To verify the repository is usable on the head node, the APT cache can be cleaned and refreshed:

```
[root@basecm11 ~]# apt clean
[root@basecm11 ~]# apt update
...
Hit:1 file:///mnt1/data/packages/HEAD-<subminor>/ubuntu/<ubuntu version> ./ InRelease
...
```

Running `apt update` only refreshes package metadata. It does not install or change any packages. This confirms the repository is being read, in the same way that `yum repolist -v` checks the YUM repositories in the preceding examples without installing anything.

Note: `apt upgrade` is deliberately not run here. Like a YUM or zypper upgrade, it upgrades every eligible package on the system by default, not just the packages served by the new repository. So running it as part of a verification step risks pulling in unrelated, and potentially breaking, package changes. It should be run for a restricted number of packages, separately and deliberately, if the repository itself is working.

To carry out the same verification on the image, `cm-chroot-sw-img` can be used to enter the image first (section 9.4.1):

```
[root@basecm11 ~]# cm-chroot-sw-img $imagedir
root@basecm11:/# apt clean
root@basecm11:/# apt update
root@basecm11:/# exit   #get out of chroot
```

9.6.4 Creating A Custom Image From The Local Repository

After having created the local repositories for SLES, RHEL or CentOS/Rocky (section 9.6.3), a custom software image based on one of these can be created. For example, for CentOS, in a directory given the arbitrary name `offlineimage`:

```
cm-create-image -d $imagedir -n offlineimage -e -s
```

The `-e` option prevents copying the default cluster manager repository files on top of the image being created, since they may have been changed by the administrator from their default status. The `-s` option prevents installing additional base distribution packages that might not be required.

9.7 Creating Images For Other Distributions And Architectures (Multidistro And Multiarch)

BCM can mix distributions in the cluster. This ability is called *multidistro*. However, it is often also loosely called *multiOS*.

BCM can also run on certain mixed architecture combinations. The ability to run on multiple architectures is called *multiarch*. For BCM, multiarch means that the node hardware can be based on either the x86-64 CPU architecture, or the ARMv8 CPU architecture, or on a mixture of both.

The Linux distributions and the hardware architectures supported by NVIDIA Base Command Manager 11.0 are shown in the following table 9.1:

		Head			
		RHEL8 (x86-64, aarch64)	RHEL9 (x86-64, aarch64)	Ubuntu 22.04, 24.04, 26.04 (x86-64, aarch64)	SLES15 (x86-64)
Image	RHEL8	x86-64, aarch64	x86-64, aarch64	x86-64, aarch64	x86-64
	RHEL9	x86-64, aarch64	x86-64, aarch64	x86-64, aarch64	x86-64
	Ubuntu 22.04	x86-64, aarch64	x86-64, aarch64	x86-64, aarch64	x86-64
	Ubuntu 24.04	x86-64, aarch64	x86-64, aarch64	x86-64, aarch64	x86-64
	Ubuntu 26.04	x86-64, aarch64	x86-64, aarch64	x86-64, aarch64	x86-64
	SLES15	x86-64	x86-64	x86-64	x86-64

Table 9.1: *Images generated by* `cm-image` *that work with head nodes, per architecture and distribution*

For example, a head node running Ubuntu 22.04, on x86-64 or ARMv8 hardware, can support Ubuntu 22.04, Ubuntu 24.04, Ubuntu 26.04, RHEL8, or RHEL9 distributions running on x86-64 and on ARMv8 hardware for the compute nodes. In addition, that same head node supports running SLES15 on the compute nodes, but only for x86-64 hardware.

The `nodearchosinfo` Command

The `nodearchosinfo` command helps the cluster administrator see what architectures and distributions are configured and reported on the nodes:

```
[root@basecm11 ~]# cmsh
```

```
[basecm11->device]% device nodearchosinfo
Hostname   Reported arch    Reported OS      Reported timestamp    Configured arch  Configured OS
---------- ---------------- ---------------- --------------------- ---------------- ----------------
basecm11   x86_64           ubuntu2204       Wed Nov 13 07:12:37   x86_64           ubuntu2204
node001    x86_64           ubuntu2204       Wed Nov 13 07:18:07   x86_64           ubuntu2204
node002    x86_64           ubuntu2204       Wed Nov 13 07:17:01   x86_64           ubuntu2204
node003    x86_64           ubuntu2204       Wed Nov 13 07:17:01   x86_64           ubuntu2204
...
```

For the reported fields, attention should be paid to the timestamps. A reported state is only updated when CMDaemon on the node reports the OS and architecture. Until that event, the reported state reports the last known value. This means that if the node fails to run after a reboot attempt is made, then the reported states are not updated, and the timestamps do not change.

The `nodearchosinfo` command also warns if the reported OS and the configured OS of the node differ.

To configure multiarch and multidistro, the `cm-image` tool is used.

9.7.1 The `cm-image` Tool

The `cm-image` tool is essentially a wrapper for the `cm-create-image` (section 9.6) tool. The `cm-image` tool however has some extra features, including allowing the cluster administrator

- to create a separate node-installer image as well as a separate software image
- to create a directory under `/cm/shared` for each image
- to select the architecture
- to manage packages in an image more easily

When used to enable a distribution for the first time, multiple changes are made to critical files and paths that may put the regular nodes into an unstable state. This means that all regular nodes should be rebooted after `cm-image` has been used. It is also recommended to set the `nextinstallmode` (page 276) to `FULL` to get rid of the original file system properly, to prevent any software leftovers from the older file system from possibly interfering with a new OS image.

The `nodearchosinfo` (page 554) command helpfully warns if the reported OS and the configured OS of the node differ.

Example

```
[basecm11->device[node001]]% set nextinstallmode <TAB><TAB>
        AUTO    FULL    MAIN    NOSYNC  SKIP
[basecm11->device[node001]]% set nextinstallmode FULL
[basecm11->device*[node001*]]% commit
[basecm11->device[node001]]%
```

The command options of `cm-image` are illustrated by the following modes and options tree:

```
cm-image
|   -h|--help
|   -v|--verbose
|   --logfile <logfile>
|---------- shell
|             -h|--help
|             -i|--image <image>
|
```

```
|---------- create
|           |----- all
|           |        -h|--help
|           |        -n, --non-interactive
|           |        -f|--force
|           |        -z|--custom-pre-install-script <custom pre-install script>
|           |        --source <archive|directory|host>
|           |        --bootstrap
|           |        --add-only
|           |        --add-archos
|           |        -b|--baserepo <base repository>
|           |        -c|--cmrepo <cluster manager repository>
|           |        -x|--excludecm <BCM packages to exclude from installation>
|           |        -j|--excludedist <distribution packages to exclude from installation>
|           |        --sles-baseurl <SLES base distribution repository URL>
|           |        --sles-extraurl <SLES extra repository URL>
|           |        --air-gapped
|           |        -a|--arch <architecture>
|           |        -d|--distro <distribution>
|           |        --dgx
|           |        --dgx-type <GPU platform type>
|           |        --no-cm-cuda-repo
|           |        --extra-pkg-group <extra packages group>
|           |        --nas-ip <NAS IP address for mounting to /cm/shared>
|           |        --nas-path <NAS directory path to mount>
|           |        --nas-filesystem <NAS filesystem type (default: NFS)>
|           |        --nas-options <NAS mount options (default: defaults)>
|           |
|           |----- node-installer
|           |        -h|--help
|           |        -n, --non-interactive
|           |        -f|--force
|           |        -z|--custom-pre-install-script <custom pre-install script>
|           |        --source <archive|directory|host>
|           |        --bootstrap
|           |        --add-only
|           |        --add-archos
|           |        -b|--baserepo <base repository>
|           |        -c|--cmrepo <cluster manager repository>
|           |        -x|--excludecm <BCM packages to exclude from installation>
|           |        -j|--excludedist <distribution packages to exclude from installation>
|           |        --sles-baseurl <SLES base distribution repository URL>
|           |        --sles-extraurl <SLES extra repository URL>
|           |        --air-gapped
|           |        -a|--arch <architecture>
|           |        -d|--distro <distribution>
|           |        --dgx
|           |        --dgx-type <GPU platform type>
|           |        --no-cm-cuda-repo
|           |        --extra-pkg-group <extra packages group>
|           |        --nas-ip <NAS IP address for mounting to /cm/shared>
|           |        --nas-path <NAS directory path to mount>
|           |        --nas-filesystem <NAS filesystem type (default: NFS)>
|           |        --nas-options <NAS mount options (default: defaults)>
|           |        --default
```

```
|           |
|           |----- swimage
|           |        -h|--help
|           |        -n, --non-interactive
|           |        -f|--force
|           |        -z|--custom-pre-install-script <custom pre-install script>
|           |        --source <archive|directory|host>
|           |        --bootstrap
|           |        --add-only
|           |        --add-archos
|           |        -b|--baserepo <base repository>
|           |        -c|--cmrepo <cluster manager repository>
|           |        -x|--excludecm <BCM packages to exclude from installation>
|           |        -j|--excludedist <distribution packages to exclude from installation>
|           |        --sles-baseurl <SLES base distribution repository URL>
|           |        --sles-extraurl <SLES extra repository URL>
|           |        --air-gapped
|           |        -a|--arch <architecture>
|           |        -d|--distro <distribution>
|           |        --dgx
|           |        --dgx-type <GPU platform type>
|           |        --no-cm-cuda-repo
|           |        --extra-pkg-group <extra packages group>
|           |        --nas-ip <NAS IP address for mounting to /cm/shared>
|           |        --nas-path <NAS directory path to mount>
|           |        --nas-filesystem <NAS filesystem type (default: NFS)>
|           |        --nas-options <NAS mount options (default: defaults)>
|           |
|           |----- fromfile <JSON input file>
|           |        -h|--help
|           |
|           '----- cmshared
|                    -h|--help
|                    -n, --non-interactive
|                    -f|--force
|                    -i|--image <image>
|                    -b|--baserepo <base repository>
|                    -c|--cmrepo <cluster manager repository>
|                    -x|--excludecm <BCM packages to exclude from installation>
|                    -j|--excludedist <distribution packages to exclude from installation>
|                    --sles-baseurl <SLES base distribution repository URL>
|                    --sles-extraurl <SLES extra repository URL>
|                    --air-gapped
|                    -a|--arch <architecture>
|                    -d|--distro <distribution>
|                    --nas-ip <NAS IP address for mounting to /cm/shared>
|                    --nas-path <NAS directory path to mount>
|                    --nas-filesystem <NAS filesystem type (default: NFS)>
|                    --nas-options <NAS mount options (default: defaults)>
|                    --default
|                    --add-only
|                    --source <ISO or DVD package source>
|
|---------- remove
|             -h|--help
```

```
|             -n|--non-interactive
|             -f|--force
|             -a|--arch <architecture>
|             -d|--distro <distribution>
|             --erase
|
'---------- package
              -h|--help
              -i|--image <image>
              --install <package(s) to install>
              --remove <package(s) to remove>
              --list
              --update <package to update>
              --update-all
              --update-cm
```

Values that can be set are:

- <*architecture*>: `aarch64`, `x86_64`
- <*distribution*>: this can be one of the distributions indicated in the following list:
  - `rhel8u0`, `rhel8u1`, `rhel8u2`, `rhel8u3`, `rhel8u4`, `rhel8u5`, `rhel8u6`, `rhel8u7`, `rhel8u8`, `rhel8u9`, `rhel8u10`
  - `rhel9u0`, `rhel9u1`, `rhel9u2`, `rhel9u3`, `rhel9u4`, `rhel9u5`, `rhel9u6`, `rhel9u7`, `rhel9u8`
  - `sles15sp1`, `sles15sp2`, `sles15sp3`, `sles15sp4`, `sles15sp5`, `sles15sp6`, `sles15sp7`
  - `ubuntu2204`, `ubuntu2404`, `ubuntu2604`

  For `rhel` in the preceding list:

  - `centos` is automatically substituted in the case of CentOS distributions
  - `rocky` is automatically substituted in the case of Rocky Linux distributions
- <*archive*>: path to a base tar file (section 9.6.1), eg:`/root/basetar/data/UBUNTU2004.tar.gz`
- <*directory*>: path to a filesystem, eg: `/root/basetar/data/untarred/`
- <*host*>: URL to a host, eg: `http://10.141.255.254/x86-iso/data/packages/dist`
- <*base repository*>: repository file for base tar, eg: `/root/bright9.2-rocky8u5-iso.repo`
- <*base distribution repository url*>: URL for base distribution repository, eg: `http://dl.rockylinux.org/$contentdir/$releasever/BaseOS/$basearch/os/`
- <*cluster manager repository*>: repository file for cluster manager, eg: `/root/cm-bright9.2-rocky8u5-iso.repo`
- <*image*>: image to operate on when managing packages, for example: `/cm/images/default-image-rhel8-aarch64` or `/cm/node-installer-centos7-x86`
- <*package to install*>: eg: `cluster-tools`
- <*package to remove*>: eg: `cluster-tools`
- <*package to update*>: eg: `cluster-tools`
- <*GPU platform type*>: `dgx_h100`, `dgx_b200`, `dgx_a100`, `dgx_gb200`, `dgx_h200`, `dgx_b300`, `dgx_gb300` (the same options as described on page 548)
- <*extra packages group*>: `doca_ofed_2.9.1-2`, `doca_ofed_2.9.1-3.0.2`, `doca_ofed_2.9.3`, `doca_ofed_2.10.0-093520`, `doca_ofed_3.0.0` (the same options as described on page 548)

9.7.2 Multidistro Examples: Provisioning From Rocky 8 Head Node To Ubuntu 24.04 Regular Nodes

Using ISO For Cluster Without Network Access

A base tar (section 9.6.1) can be used

```
[root@basecm11 ~]# module load cm-image
[root@basecm11 ~]# cm-image --verbose create all -a x86_64 -d ubuntu2404 --source \
 /root/basetar/data/UBUNTU2404.tar.gz
Creating software image default-image-ubuntu2404-x86_64
...
Do you want to continue?(y/n): y
...
Creating software image default-image-ubuntu2404-x86_64.....[  OK  ]
...
Creating node installer image.....[  OK  ]
...
Creating cm-shared image at /cm/shared-ubuntu2404-x86_64...
Unmounting /cm/images/default-image-ubuntu2404-x86_64/cm/shared
...
Adding cm-shared image entities.....[  OK  ]
...
Updating cmd entities
Changing fspart /cm/shared to /cm/shared-rocky8-x86_64
Changing fspart /cm/node-installer to /cm/node-installer-rocky8-x86_64
...
Creating ramdisk.....[  OK  ]
Added new category: default-ubuntu2404-x86_64
Use this category for adding nodes
Completed
```

As suggested by the output, a new category, `default-ubuntu2404-x86_64`, appears.
A node can be placed in the new category and restarted:

```
[root@basecm11 ~]# cmsh
[basecm11]% device use node001
[basecm11->device[node001]]% set category <TAB><TAB>
default-rocky8-x86_64    default-ubuntu2404-x86_64
[basecm11->device*[node001*]]% commit
[basecm11->device[node001]]%

...15:30:57 2024 [notice] basecm11: node001 [   UP   ], restart required (category)
[basecm11->device[node001]]%
...15:31:06 2024 [notice] basecm11: Service dhcpd was restarted
[basecm11->device[node001]]% set nextinstallmode FULL
[basecm11->device*[node001*]]% commit
[basecm11->device[node001]]% reboot
node001: Reboot in progress ...
```

Adding Several Update Versions Alongside Each Other

In the following session, a Rocky9u3 x86 archived base tar is being added with `cm-image`. A Rocky9u2 software image is then being added with `cm-create-image`, using the same node-installer and `/cm/shared/` images directory:

```
[root@basecm11 ~]# module load cm-image
[root@basecm11 ~]# cm-image --verbose create all -a x86_64 -d rocky9u3 --source \
 /root/basetar/data/ROCKY9u3.tar.gz
```

```
[root@basecm11 ~]# cm-create-image -a /run/ROCKY9u2.tar.gz -f -n \
default-image-rocky9u2-x86_64 -i /cm/images/default-image-rocky9u2-x86_64 -g public
[root@basecm11 ~]# cmsh
[basecm11]% device use node001
[basecm11->device[node001]]% set category default-rocky9u3-x86_64
[basecm11->device[node001]]% set softwareimage default-rocky9u2-x86_64
[basecm11->device[node001]]% set nextinstallmode FULL
[basecm11->device*[node001*]]% commit
```

Configuring CMDaemon With The Software Image, Node-installer, Or Shared Filesystem

The `cm-image` tool can be used to add a CMDaemon configuration for a software image, node-installer, or shared filesystem to a CMDaemon database.

If a `cm-image`-generated image directory, for example `default-image-ubuntu2404-x86_64`, has been copied over from another cluster:

```
[root@basecm11 images]# pwd; ls -l
/cm/images
total 0
dr-xr-xr-x 21 root root 295 Mar 20 23:47 default-image
drwxr-xr-x 23 root root 247 Mar 25 12:56 default-image-ubuntu2404-x86_64
```

then the local CMDaemon can be configured for it with the `--add-only` option:

```
[root@basecm11 ~]# module load cm-image
[root@basecm11 ~]# cm-image create swimage -a x86_64 -d ubuntu2404 --add-only
```

9.7.3 Multiarch Example: Creating An Image From A Centos 8 Head Node For ARMv8 Architecture Regular Nodes

This section explains how to configure a software image for a regular node that runs on ARMv8 hardware, assuming BCM is installed on a head node (Chapter 3 of the *Installation Manual*).

Assuming an ARMv8 ISO `bright91-rhel8u2.aarch64.iso` has been picked up, it can be mounted for web access with:

```
[root@basecm11 ~]# mkdir /var/www/html/aarch64-iso
[root@basecm11 ~]# mount -o loop /root/bright91-rhel8u2.aarch64.iso /var/www/html/aarch64-iso
```

A repository file can be created with the following content:

```
[root@basecm11 ~]# cat /root/rhel8-aarch64-cm-iso.repo
[dist-packages-rhel8-aarch64]
name=Dist packages rhel8 aarch64
baseurl=http://10.141.255.254/aarch64-iso/data/packages/dist
enabled=1
gpgcheck=0

[cm-packages-rhel8-aarch64]
name=CM packages rhel8 aarch64
baseurl=http://10.141.255.254/aarch64-iso/data/packages/9.1
enabled=1
gpgcheck=0

[cm-packages-rhel8-aarch64-hpc]
name=CM packages rhel8 aarch64 HPC
baseurl=http://10.141.255.254/aarch64-iso/data/packages/packagegroups/hpc
enabled=1
gpgcheck=0
```

This assumes that the head node has the IP address 10.141.255.254. It should be changed if needed.

It also assumes the HEAD packages are in the HEAD packages directory of the ISO. If it is not, then the corresponding `baseurl` string should be changed if needed. Thus, if, for example, after inspecting the loop-mounted paths under `/var/www/html/`, the relative path `data/packages/9.1` has changed to `data/packages/9.1-6`, then the `baseurl` should be changed to end in `9.1-6` instead of `9.1` too.

The images can then be created with:

```
[root@basecm11 ~]# module load cm-image
[root@basecm11 ~]# cm-image --verbose create all -a aarch64 -d rhel8 --source \
/var/www/html/aarch64-iso/data/RHEL8u2.tar.gz -b /root/rhel8-aarch64-cm-iso.repo -c \
/root/rhel8-aarch64-cm-iso.repo
```

This takes a while to complete. At the end of the process the following ARMv8 images and entities are created by default:

- The node-installer image: `/cm/node-installer-rhel8-aarch64`
- The `/cm/shared-...` directory: `/cm/shared-rhel8-aarch64`
- The node image: `/cm/images/default-image-rhel8-aarch64`
- The node category: `default-rhel8-aarch64`

The preceding can be verified via `cmsh`:

```
[root@basecm11 ~]# cmsh
[basecm11]% category list
Name (key)               Software image                        Nodes
------------------------ ------------------------------------- --------
default-centos8-x86_64   default-image                         1
default-rhel8-aarch64    default-image-rhel8-aarch64           1
[basecm11]% softwareimage list
Name (key)                  Path                                  ...
--------------------------- ------------------------------------- ...
default-image               /cm/images/default-image              ...
default-image-rhel8-aarch64 /cm/images/default-image-rhel8-aarch64 ...
[basecm11]% partition archos base; list
Arch    OS    Primary image               Shared                    Installer
------- ----- --------------------------- ------------------------- --------------------------------
x86_64  rhel8 default-image               /cm/shared-centos8-x86_64 /cm/node-installer-centos8-x86_64
aarch64 rhel8 default-image-rhel8-aarch64 /cm/shared-rhel8-aarch64  /cm/node-installer-rhel8-aarch64
```

The node settings should be updated. The new category can be assigned to any ARMv8 nodes:

```
[root@basecm11 ~]# cmsh
[basecm11]% device use arm-node001
[basecm11->device[node001]]% set category default-rhel8-aarch64
[basecm11->device*[node001*]]% commit
```

Carrying out changes to `primaryimage` requires an associated category: The value defined for the property `primaryimage` decides the software image used to boot new nodes. The image also tracks what packages are used under its associated shared directory, via the RPM or APT database. The image for `primaryimage` and the associated shared directory can be set with `cmsh` from within the `archos` submode, under the top-level `partition` mode.

Example

```
[basecm11->partition[base]->archos]% list
Arch    OS           Primary image                     Shared                          Installer
------- ------------ --------------------------------- ------------------------------- ---------
aarch64 ubuntu1804   default-image-ubuntu1804-aarch64  /cm/shared-ubuntu1804-aarch64  /cm/n....
x86_64  rhel8        default-image                     /cm/shared-centos8-x86_64      /cm/n....
[basecm11->partition[base]->archos]% use aarch64/ubuntu1804
[basecm11->partition[base]->archos[aarch64/ubuntu1804]]% set primaryimage
default-image       default-image-ubuntu1804-aarch64  new-image-ubuntu1804-aarch64
[basecm11->...->archos[aarch64/ubuntu1804]]% set primaryimage new-image-ubuntu1804-aarch64
[basecm11->partition*[base*]->archos*[aarch64/ubuntu1804*]]% commit
```

If `cm-image` is used to generate a new architecture and operating system, then the primary image is automatically set. Otherwise, by default, the value of `primaryimage` is not set.

If the value of `primaryimage` is set, then it is strongly recommended that a category that has that image must exist.

If such a category does not exist, then CMDaemon uses the RPM or APT database of the new image to decide what the packages are on the shared directory.

If the value of `primaryimage` is set and multiple categories use the image, then the first category that is found is used.

Setting the `bootloaderprotocol` for ARMv8 hardware: The `bootloaderprotocol` should be set to `tftp` to work with ARMv8 hardware:

Example

```
[root@basecm11 ~]# cmsh
[basecm11]% category use default-rhel8-aarch64
[basecm11->category[default-rhel8-aarch64]]% set bootloaderprotocol tftp
[basecm11->category*[default-rhel8-aarch64*]]% commit
```

Setting the `kernelconsoleoutput` for ARMv8 hardware: The `kernelconsoleoutput` should be changed to `ttyAMA0` to work with the image running on the ARMv8 hardware:

```
[root@basecm11 ~]# cmsh
[basecm11]% softwareimage use default-rhel8-aarch64
[basecm11->softwareimage[default-rhel8-aarch64]]% set kerneloutputconsole ttyAMA0
[basecm11->softwareimage*[default-rhel8-aarch64*]]% commit
```

The settings configured so far are for generic ARMv8 hardware.

Fujitsu ARMv8 Hardware Configuration

Nodes using Fujitsu ARMv8 hardware can have their configuration options modified further.

The BMC settings of the nodes should be updated with extra arguments:

Example

```
[root@basecm11 ~]# cmsh
[basecm11]% device use arm-node001
[basecm11->device[node001]]% bmcsettings; set extraarguments "-L USER -t 0x30"
[basecm11->device*[node001*]]% commit
```

The configuration for fetching environmental metrics should also be updated. The `ipmitool` monitoring resource available for Fujitsu ARMv8 hardware is run via the `a64fx` resource.

The existence of the `a64fx` monitoring resource can be checked for on the ARMv8 node:

```
[root@basecm11 ~]# cmsh
[basecm11]% device monitoringresources arm-node001 | grep a64fx
a64fx
```

The monitoring settings for IPMI via the `a64fx` object can be enabled as follows:

```
[root@basecm11 ~]# cmsh
[basecm11]% monitoring setup; use ipmi
[basecm11->monitoring->setup[ipmi]]% set script /cm/local/apps/cmd/scripts/metrics/sample_ipmitool.py
[basecm11->monitoring->setup*[ipmi*]]% executionmultiplexers
[basecm11->monitoring->setup*[ipmi*]->executionmultiplexers]% remove ipmi
[basecm11->monitoring->setup*[ipmi*]->executionmultiplexers*]% add resource a64fx
[basecm11->monitoring->setup*[ipmi*]->executionmultiplexers*[a64fx*]]% set resources a64fx
[basecm11->monitoring->setup*[ipmi*]->executionmultiplexers*[a64fx*]]% commit
```

10

Monitoring: Monitoring Cluster Devices

BCM monitoring allows a cluster administrator to monitor anything that can be monitored in the cluster. Much of the monitoring consists of pre-defined sampling configurations. If there is anything that is not configured, but the data on which it is based can be sampled, then monitoring can be configured for it too, by the administrator.

The monitoring data can be viewed historically, as well as on demand. The historical monitoring data can be stored raw, and optionally also as consolidated data—a way of summarizing data.

The data can be handled raw and processed externally, or it can be visualized within Base View in the form of customizable charts. Visualization helps the administrator spot trends and abnormal behavior, and is helpful in providing summary reports for managers.

Monitoring can be configured to set off alerts based on triggers, and pre-defined or custom actions can be carried out automatically, depending on triggers. The triggers can be customized according to user-defined conditional expressions.

Carrying out such actions automatically after having set up triggers for them means that the monitoring system can free the administrator from having to carry out these chores.

In this chapter, the monitoring system is explained with the following approach:

1. A basic example is first presented in which processes are run on a node. These processes are monitored, and trigger an action when a threshold is exceeded.

2. With this easy-to-understand example as a basic model, the various features and associated functionality of the BCM monitoring system are then described and discussed in further depth. These include visualization of data, concepts, configuration, monitoring customization and `cmsh` use.

10.1 A Basic Monitoring Example And Action

10.1.1 Synopsis Of Basic Monitoring Example

In section 10.1, after an overview (section 10.1.1), a minimal basic example of monitoring a process is set up (section 10.1.2) and used (section 10.1.3). The example is contrived, with the aim being to present a basic example that covers a part of what the monitoring system is capable of handling. The basic example gives the reader a structure to keep in mind, around which further details are fitted and filled in during the coverage in the rest of this chapter.

In the basic example, a user runs a large number of pointless CPU-intensive processes on a head node which is normally very lightly loaded. An administrator who is monitoring user mode CPU load usage throughout the cluster, notices this usage spike. After getting the user to stop wasting CPU cycles, the administrator may decide that putting a stop to such processes automatically is a good idea. The administrator can set that up with an action that is triggered when a high load is detected. The action that is taken after triggering, is to stop the processes (figure 10.1).

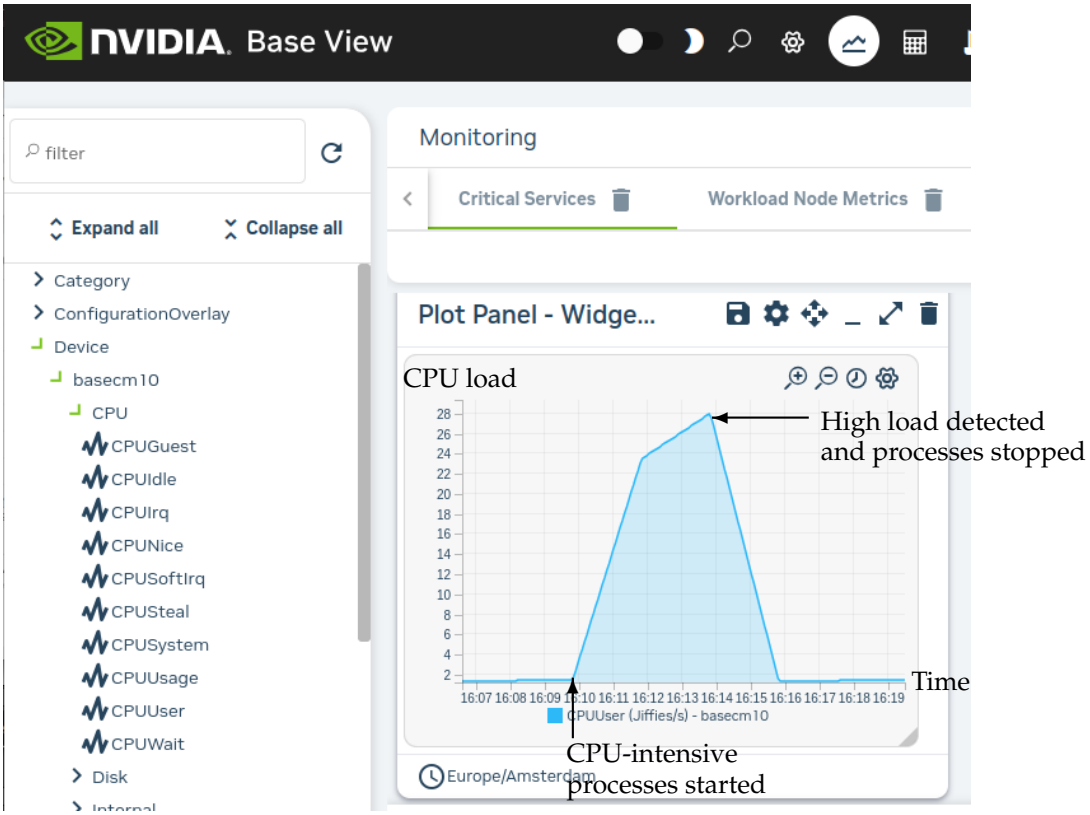


Figure 10.1: Monitoring Basic Example: CPU-intensive Processes Started, Detected And Stopped

The basic example thus illustrates how BCM monitoring can be used to detect something on the cluster and how an action can be set up and triggered based on that detection.

10.1.2 Before Using The Basic Monitoring Example—Setting Up The Pieces

Running A Large Number Of Pointless CPU-Intensive Processes

One way to simulate a user running pointless CPU-intensive processes is to run several instances of the standard unix utility, `yes`. The `yes` command sends out an endless number of lines of "`y`" texts. It is typically used in scripts to answer dialog prompts for confirmation.

The administrator can run 8 subshell processes in the background from the command line on the head node, with `yes` output sent to `/dev/null`, as follows:

```
for i in {1..8}; do ( yes > /dev/null &); done
```

Running "`mpstat 2`" shows usage statistics for each processor, updating every 2 seconds. It shows that `%usr`, which is user mode CPU usage percentage, is close to 90% on an 8-core or less head node when the 8 subshell processes are running.

Setting Up The Kill Action

To stop the pointless CPU-intensive `yes` processes, the command `killall yes` can be used. The administrator can make it a part of a script `killallyes`:

```
#!/bin/bash
killall yes
```

and make the script executable with a `chmod 700 killallyes`. For convenience, it may be placed in the `/cm/local/apps/cmd/scripts/actions` directory where some other action scripts also reside.

10.1.3 Using The Basic Monitoring Example

Now that the pieces are in place, the administrator can use Base View to add the `killallyesaction` action to its action list, and then set up a trigger for the action:

Adding The Action To The Actions List

In Base View:

- The navigation path

  `Monitoring > Actions > Monitoring Actions > killprocess: > Clone`

  is used to clone the object structure of an existing action. The `killprocess` action is a convenient choice of existing action to clone because it is expected to function in a similar way to `killallyes`, which means that the values in the cloned object should not have to be modified much. However, any action could be cloned and the clone modified in appropriate places.

- The name of the cloned action is changed. That is, the administrator sets `Name` to `killallyesaction`. This is just a sensible label—the name can be arbitrary.

- `Script` is set to the path `/cm/local/apps/cmd/scripts/actions/killallyes`, which is where the script was placed earlier (page 566).

After saving, the `killallyesaction` action becomes part of the list of monitoring actions (figure 10.2):

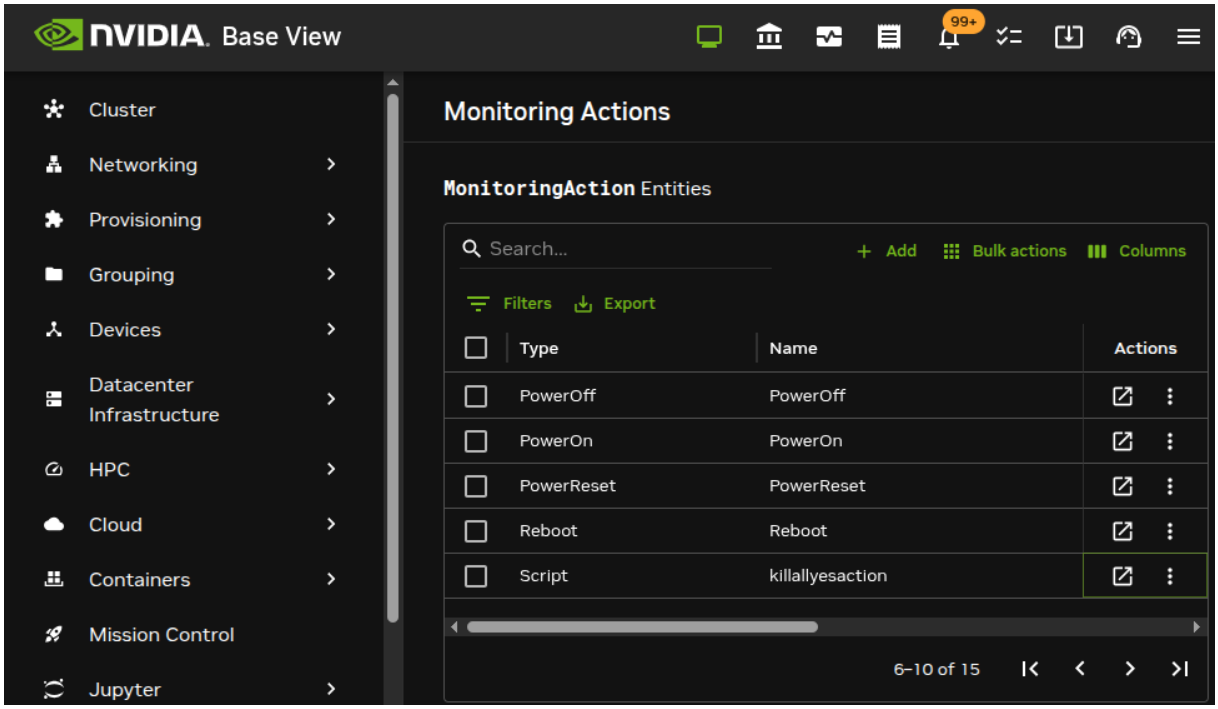


Figure 10.2: Base View Monitoring Configuration: Adding An Action

Setting Up A Trigger Using CPUUser On The Head Node(s)

The navigation path

`Monitoring > Triggers > Failing health checks > Clone`

can be used to configure a monitoring trigger, by cloning an existing trigger. A trigger is a condition

that is set on the state of sample, which runs an action when that condition is met. In this case, the sample state condition may be that the metric (section 10.2.3) `CPUUser` must not exceed 50. If it does, then an action (`killallyesaction`) is run, which should kill the `yes` processes.

- `CPUUser` is a measure of the time spent in user mode CPU usage per second, and is measured in jiffy intervals per second.
- A jiffy interval is a somewhat arbitrary time interval that is predefined for kernel developers per platform. It is the minimum amount of time that a process has access to the CPU before the kernel can switch to another task.

  The jiffy interval per second of `CPUUser` is a quantity rather than a percentage. It should not be confused with the closely related measurable `CPUUsage`, which is a percentage. `CPUUsage` is used for `%user` monitoring, where `%user` is the user time as defined and measured for the `top` command.

To configure triggering for `CPUUser`, the trigger attributes can be modified as follows (figure 10.3):

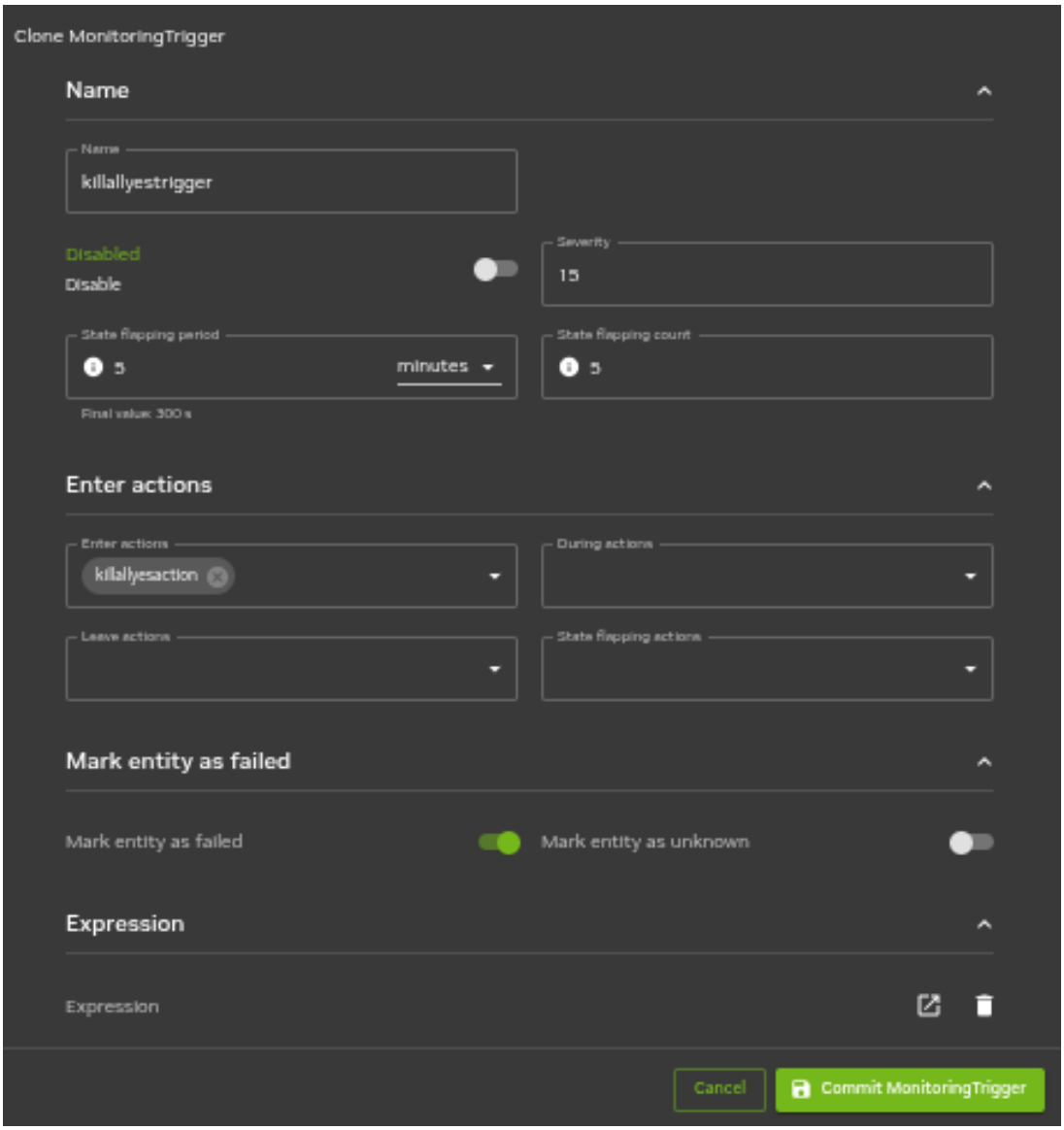


Figure 10.3: Base View Monitoring Configuration: Setting A Trigger

- A name is set for the trigger. The name can be arbitrary, and `killallyestrigger` is used in this example.
- The trigger is enabled (that is: the `Disabled` toggle is not enabled).
- The `Enter actions` field is filled with the `killallyesaction`, which is the action defined earlier.
- The trigger is saved by clicking on the `Commit MonitoringTrigger` button.
- the trigger can be set to run an action script if the sample state crosses over into a state that meets the trigger condition. That is, `Enter actions` is configured for a particular condition.

  The condition under which the `Enter actions` action script is run in the example, can simply be when `CPUUser` on the head node is above 50. Such a condition can be set by setting an expression in a subwindow. The subwindow to do this is opened with the `Expression` box icon seen at the bottom of the screen in figure 10.3.

  Clicking on the icon brings up the `MonitoringCompareExpression` subwindow (figure 10.4):

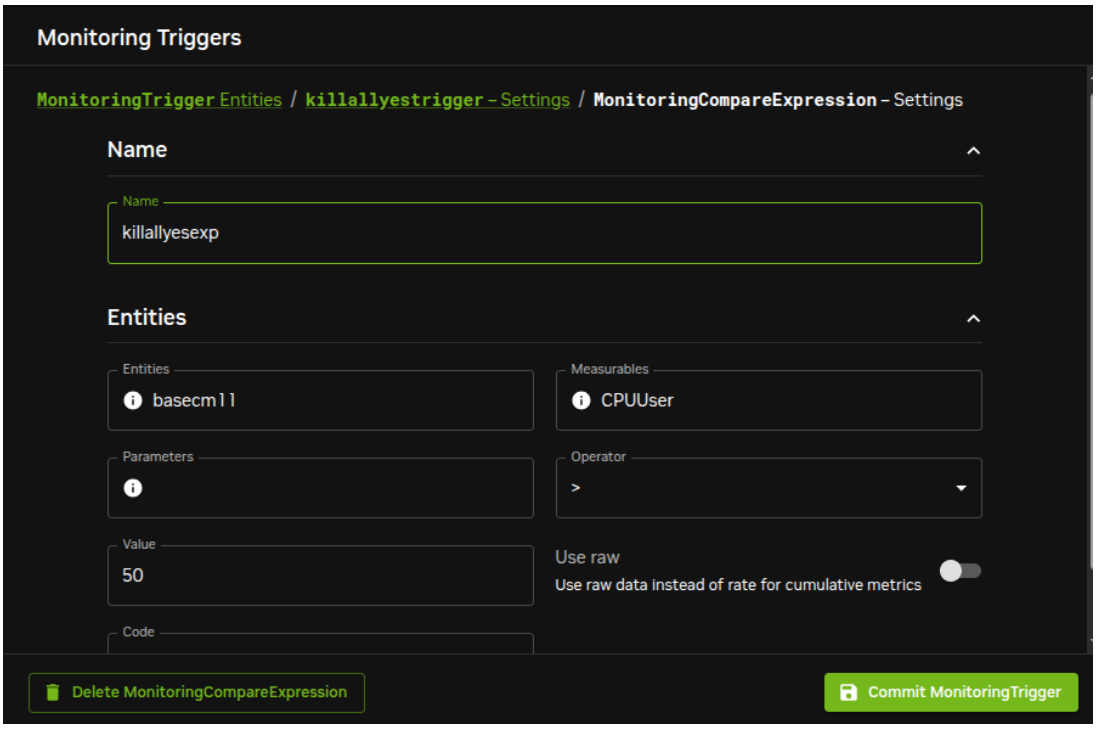


Figure 10.4: Base View Monitoring Configuration: Setting An Expression

Within the subwindow:

- A name is set for the expression. The name can be arbitrary, and `killallyesexp` is used for `Name` in this example.
- An entity is set. In this case, the entity being monitored is the head node. If the head node is called `basecm11` in this example, then `basecm11` is the value set for `Entities`. An entity is often simply a device, but it can be any object that CMDaemon stores.
- A measurable is set. In this case, `Measurables` is set to `CPUUser`.
- An operator and threshold value are set. In this case `>`, which is the greater than operator, and `50` which is a significant amount of CPUUser time in jiffies/s, are set for `Operator` and `Value`.

After saving the configuration, the killallyesexp expression evaluates the data being sampled for the killallyestrigger trigger. If the expression is TRUE, then the trigger launches the killallyesaction action.

The Result

In the preceding section, an action was added, and a trigger was set up with a monitoring expression.

With a default installation on a newly installed cluster, the measurement of CPUUser is done every 120s (the period can be modified in the Data Producer window of Base View, as seen in figure 10.11). The basic example configured with the defaults thus monitors if CPUUser on the head node has crossed the bound of 50 jiffies/s every 120s.

If CPUUser is found to have entered—that is: crossed over from below the value and gone into the zone beyond 50 jiffies/s—then the killallyesexp expression notices that. Then, the trigger it is configured for, killallyestrigger trigger, runs the killallyesaction action, which runs the killallyes script. The killallyes script kills all the running yes processes. Assuming the system is trivially loaded apart from these yes processes, the CPUUser metric value then drops to below 50 jiffies/s.

To clarify what "found to have entered" means in the previous paragraph:

After an Enter trigger condition has been met for a sample, the first sample immediately after that does not ever meet the Enter trigger condition, because an Enter threshold crossing condition requires the previous sample to be below the threshold.

The second sample can only launch an action if the Enter trigger condition is met and if the preceding sample is below the threshold.

Other non-yes CPU-intensive processes running on the head node can also trigger the killallyes script. Since the script only kills yes processes, leaving any non-yes processes alone, it would in such a case run unnecessarily. This is a deficiency due to the contrived and simple nature of the basic example which is being illustrated here. In a production case the action script is expected to have a more sophisticated design.

At this point, having gone through section 10.1, the reader is expected to have a rough idea of how monitoring, triggers, trigger conditional expressions, and actions work. The following sections in this chapter cover the concepts and features for BCM monitoring in greater detail.

10.2 Monitoring Concepts And Definitions

A discussion of the concepts of monitoring, along with definitions of terms used, is appropriate at this point. The features of the monitoring system in BCM covered later on in this chapter will then be understood more clearly.

10.2.1 Measurables

Measurables are measurements (sample values) that are obtained via data producers (section 10.2.10) in CMDaemon's monitoring system. The measurements can be made for nodes, head nodes, other devices, or other *entities*.

Types Of Measurables

Measurables can be:

- *enummetrics*: measurements with a small number of states. The states can be pre-defined, or user-defined. Further details on enummetrics are given in section 10.2.2.
- *metrics*: measurements with number values, and no data, as possible values. For example, values such as: -13113143234.5, 24, 9234131299. Further details on metrics are given in section 10.2.3.
- *health checks*: measurements with the states PASS, FAIL, and UNKNOWN as possible states, and no data as another possible state, when none of the other states are set. Further details on health checks are given in section 10.2.4.

no data And Measurables

If no measurements are carried out, but a sample value needs to be saved, then the sample value is set to no data for a measurable. This is a defined value, not a null data value. metrics and enummetrics can therefore also take the no data value.

Entities And Measurables

An entity is a concept introduced in BCM version 8.0.

Normally, a device, or a category or some similar grouping is a convenient idea to keep in mind as an entity, for concreteness.

The default entities in a new installation of BCM are the following:

device category partition[base] softwareimages

However, more generally, an entity can be an object from the following modes of cmsh:

category cloud configurationoverlay device edgesite etcd fspart group jobqueue jobs kubernetes network nodegroup partition profile rack softwareimage user

For example, a software image object that is to be provisioned to a node is an entity, with some of the possible attributes of the entity being the name, kernelversion, creationtime, or locked attributes of the image:

```
[root@basecm11 ~]# cmsh -c "softwareimage use default-image; show"
Parameter                      Value
------------------------------ ------------------------------------------------
Creation time                  Thu, 08 Jun 2017 18:15:13 CEST
Enable SOL                     no
Kernel modules                 <44 in submode>
Kernel parameters
Kernel version                 3.10.0-327.3.1.el7.x86_64
Locked                         no
Name                           default-image
...
```

Because measurements can be carried out on such a variety of entities, it means that the monitoring and conditional actions that can be carried out on a BCM cluster can be very diverse. This makes entities a powerful and versatile concept in BCM's monitoring system for managing clusters.

Listing Measurables Used By An Entity

In cmsh, for an entity, such as a device within device mode, a list of the measurables used by that device can be viewed with the measurables command.

Example

```
[basecm11->device]% measurables node001
Type         Name                Parameter  Class     Producer
------------ ------------------- ---------- --------- ---------------
Enum         DeviceStatus                   Internal  DeviceState
HealthCheck  ManagedServicesOk              Internal  CMDaemonState
HealthCheck  defaultgateway                 Network   defaultgateway
HealthCheck  diskspace                      Disk      diskspace
HealthCheck  dmesg                          OS        dmesg
...
...
```

The subsets of these measurables—enummetrics, metrics, and health checks—can be listed with the enummetrics (section 10.2.2), metrics (section 10.2.3), or healthchecks (section 10.2.4) command.

In Base View, all the entities that are using health checks can be viewed via the navigation path:

Monitoring > All Health Checks (figure 10.25 section 10.4.7)

Listing Entities That Use A Measurable

The entities using a specific measurable can be listed with the usage command:

Example

```
[basecm11->monitoring->measurable]% usage nfs_v3_server_total
Measurable           Count  Entities
-------------------- ------ --------------
nfs_v3_server_total  1      basecm11
```

If the number of measurables is too large to view on the screen:

Example

```
[basecm11->monitoring->measurable]% usage devicestatus
Measurable    Count  Entities
------------ ------ ------------------------------------------------------------------------------
DeviceStatus  21     node001,node002,node003,node004,node005,node006,node007,node008,node009,node010+
```

then the -v option can be used to list the entities over multiple lines:

Example

```
basecm11->monitoring->measurable]% usage -v devicestatus
Measurable    Count  Entities
------------ ------ ------------------------------------------------------------------------------
DeviceStatus  21     node001,node002,node003,node004,node005,node006,node007,node008,node009,node010,
                     node011,node012,node013,node014,node015,node016,node017,node018,node019,node020,
                     basecm11
```

Listing Measurables From monitoring Mode

Similarly, under monitoring mode, within the measurable submode, the list of measurable objects that can be used can be viewed with a list command:

Example

```
[basecm11->monitoring]% measurable list
Type         Name (key)          Parameter  Class                        Producer
------------ ------------------- ---------- ---------------------------- ------------------
Enum         DeviceStatus                   Internal                     DeviceState
HealthCheck  ManagedServicesOk              Internal                     CMDaemonState
HealthCheck  Mon::Storage                   Internal/Monitoring/Storage  MonitoringSystem
HealthCheck  chrootprocess                  OS                           chrootprocess
HealthCheck  cmsh                           Internal                     cmsh
...
...
```

The subsets of these measurables—enummetrics, metrics, and health checks—can be listed with: list enum (section 10.2.2), list metric (section 10.2.3), or list healthcheck (section 10.2.4).

In Base View, the equivalent to listing the measurables can be carried out via the navigation path:

Monitoring > Measurables (figure 10.12, section 10.4.2)

Viewing Parameters For A Particular Measurable From `monitoring` Mode

Within the `measurable` submode, parameters for a particular measurable can be viewed with the `show` command for that particular measurable:

Example

```
[basecm11->monitoring->measurable]% use devicestatus
[basecm11->monitoring->measurable[DeviceStatus]]% show
Parameter                      Value
------------------------------ ----------------------
Class                          Internal
Consolidator                   none
Description                    The device status
Disabled                       no (DeviceState)
Maximal age                    0s (DeviceState)
Maximal samples                4,096 (DeviceState)
Name                           DeviceStatus
Parameter
Producer                       DeviceState
Revision
Type                           Enum
```

10.2.2 Enummetrics

An *enummetric* is a measurable for an entity that can only take a limited set of values. At the time of writing of this section (February 2026), the enummetrics, and their possible values, are:

The values possible for the enummetrics are as follows:

- `DeviceStatus`:

  `up`, `down`, `closed`, `installing`, `installer_failed`, `installer_rebooting`, `installer_calling_init`, `installer_unreachable`, `installer_burning`, `burning`, `unknown`, `opening`, `going_down`, `pending`, and `no data`

- `wlm_slurm_state`:

  `allocated`, `completing`, `down`, `drain`, `draining`, `fail`, `failing`, `idle`, `maint`, and `mixed`

- `sys_class_net_name_assign_type`:

  `unpredictable`, `predictable`, `userspace`, and `renamed`

The list of possible values for an enummetric can be seen with the `enumvalues` command, used within the enummetric object:

Example

```
root@basecm11:~# cmsh -c "monitoring measurable; enumvalues devicestatus"
Index    Value
-------- ----------------------
0        up
1        down
...
```

The enummetrics available for use can be listed from within the `measurable` submode of the `monitoring` mode:

Example

```
[basecm11->monitoring->measurable]% list enum
Type   Name (key)               Parameter           Class     Producer
------ ----------------------- ------------------- --------- --------------------
Enum   DeviceStatus                                Internal  DeviceState
Enum   wlm_slurm_state                             Workload  slurm-state-count
[basecm11->monitoring->measurable]%
```

The list of enummetrics that is configured to be used by an entity, such as a device, can be viewed with the `enummetrics` command for that entity:

Example

```
[basecm11->device]% enummetrics node001
Type   Name                    Parameter           Class     Producer
------ ----------------------- ------------------- --------- -------------------
Enum   DeviceStatus                                Internal  DeviceState
[basecm11->device]%
```

The states that the entity has been through can be viewed with a `dumpmonitoringdata` command (section 10.6.4):

Example

```
[basecm11->device]% dumpmonitoringdata -99d now devicestatus node001
Timestamp                  Value       Info
-------------------------- ----------- ----------
2017/07/03 16:07:00.001    down
2017/07/03 16:09:00.001    installing
2017/07/03 16:09:29.655    no data
2017/07/03 16:11:00        up
2017/07/12 16:05:00        up
```

The parameters of an enummetric such as `devicestatus` can be viewed and set from `monitoring` mode, from within the `measurable` submode (page 573).

10.2.3 Metrics

A *metric* for an entity is typically a numeric value for an entity. The value can have units associated with it.

In the basic example of section 10.1, the metric value considered was `CPUUser`, measured at the default regular time intervals of 120s.

The value can also be defined as `no data`. `no data` is substituted for a null value when there is no response for a sample. `no data` is not a null value once it has been set. This means that there are no null values stored for monitored data.

Other examples for metrics are:

- `LoadOne` (value is a number, for example: 1.23)
- `WriteTime` (value in ms/s, for example: 5 ms/s)
- `MemoryFree` (value in readable units, for example: 930 MiB, or 10.0 GiB)

A metric can be a built-in, which means it comes with BCM as integrated code within CMDaemon. This is based on `c++` and is therefore much faster than the alternative. The alternative is that a metric can be a standalone script, which means that it typically can be modified more easily by an administrator with scripting skills.

The word metric is often used to mean the script or object associated with a metric as well as a metric value. The context makes it clear which is meant.

A list of metrics in use can be viewed in `cmsh` using the `list` command from `monitoring` mode:

Example

```
[basecm11->monitoring]% measurable list metric
Type    Name (key)              Parameter      Class              Producer
------- ----------------------- -------------- ------------------ -------------
Metric  AlertLevel              count          Internal           AlertLevel
Metric  AlertLevel              maximum        Internal           AlertLevel
...
```

In Base View, the metrics can be viewed with the navigation path:

`Monitoring` > `Measurables` (figure 10.12, section 10.4.2)

A list of metrics in use by an entity can be viewed in `cmsh` using the `metrics` command for that entity. For example, for the entity `node001` in mode `devices`:

Example

```
[basecm11->devices]% metrics node001
Type    Name                    Parameter      Class     Producer
------- ----------------------- -------------- --------- --------------
Metric  AlertLevel              count          Internal  AlertLevel
Metric  AlertLevel              maximum        Internal  AlertLevel
...
```

The parameters of a metric such as `AlertLevel:count` can be viewed and set from `monitoring` mode, from within the `measurable` submode, just as for the other measurables:

Example

```
[basecm11->monitoring->measurable]% use alertlevel:count
[basecm11->monitoring->measurable[AlertLevel:count]]% show
Parameter                      Value
------------------------------ ----------------------
Class                          Internal
Consolidator                   default
Cumulative                     no
Description                    Number of active triggers
Disabled                       no
Maximal age                    0s
Maximal samples                0
Maximum                        0
Minimum                        0
Name                           AlertLevel
Parameter                      count
Producer                       AlertLevel
Revision
Type                           Metric
```

The equivalent Base View navigation path to edit the parameters is:

`Monitoring` > `Measurables` >`Edit`

10.2.4 Health Check

A *health check* value is a response to a check carried out on an entity. The response indicates the health of the entity for the check that is being carried out.

For example, the `ssh2node` health check, which runs on the head node to check if the SSH port 22 passwordless access to regular nodes is reachable.

A health check is run at a regular time interval, and can have the following possible values:

- `PASS`: The health check succeeded. For example, if `ssh2node` is successful, which suggests that an `ssh` connection to the node is fine.
- `FAIL`: The health check failed. For example, if `ssh2node` was rejected. This suggests that the `ssh` connection to the node is failing.
- `UNKNOWN`: The health check did not succeed, did not fail, but had an unknown response. For example, if `ssh2node` has a timeout, for example due to routing or other issues. It means that it is unknown whether the connection is fine or failing, because the response that came in is unknown. Typically the administrator should investigate this further.
- `no data`: The health check did not run, so no data was obtained. For example, if `ssh2node` is disabled for some time, then no data values were obtained during this time. Since the health check is disabled, it means that `no data` cannot be recorded during this time by `ssh2node`. However, because having a `no data` value in the monitoring data for this situation is a good idea—explicitly knowing about having no data is helpful for various reasons—then `no data` values can be set, by CMDaemon, for samples that have no data.

Other examples of health checks are:

- `diskspace`: check if the hard drive still has enough space left on it
- `mounts`: check mounts are accessible
- `mysql`: check status and configuration of MySQL is correct
- `hpraid`: check RAID and health status for certain HP RAID hardware

These and others can be seen in the directory: `/cm/local/apps/cmd/scripts/healthchecks`.

Health Checks

In Base View, the health checks that can be configured for all entities can be seen with the navigation path:

`Monitoring` > `Measurables` (figure 10.12, section 10.4.2)

Options can be set for each health check by clicking through via the `Edit` button.

All Configured Health Checks

In Base View, health checks that have been configured for all entities can be seen with the navigation path:

`Monitoring` > `All Health Checks` (section 10.4.7)

The view can be filtered per column.

Configured Health Checks For An Entity

An overview can be seen for a particular entity <*entity*> via the navigation path:

`Monitoring` > `Health status`><*entity*>>`Show`

Severity Levels For Health Checks, And Overriding Them

A health check has a settable severity (section 10.2.7) associated with its response defined in the trigger options.

For standalone healthchecks, the severity level defined by the script overrides the value in the trigger. For example, `FAIL 40` or `UNKNOWN 10`, as is set in the `hpraid` health check (`/cm/local/apps/cmd/scripts/healthchecks/hpraid`).

Severity values are processed for the `AlertLevel` metric (section 10.2.8) when the health check runs.

Default Templates For Health Checks And Triggers

A health check can also launch an action based on any of the response values.

Monitoring triggers have the following default templates:

- `Failing health checks`: With a default severity of `15`
- `Passing health checks`: With a default severity of `0`
- `Unknown health checks`: With a default severity of `10`

The severity level is one of the default parameters for the corresponding health checks. These defaults can also be modified to allow an action to be launched when the trigger runs, for example, sending an e-mail notification whenever any health check fails.

With the default templates, the actions are by default set for all health checks. However, specific actions that are launched for a particular measurable instead of for all health checks can be configured. To do this , one of the templates can be cloned, the trigger can be renamed, and an action can be set to launch from a trigger. The reader should be able to recognize that in the basic example of section 10.1 this is how, when the metric measurable `CPUUser` crosses 50 jiffies/s, the `killallyestrigger` is activated, and the `killallyes` action script is run.

10.2.5 Trigger

A *trigger* is a threshold condition set for a sampled measurable. When a sample crosses the threshold condition, it enters or leaves a zone that is demarcated by the threshold.

A trigger zone also has a settable severity (section 10.2.7) associated with it. This value is processed for the `AlertLevel` metric (section 10.2.8) when an action is triggered by a threshold event.

Triggers are discussed further in section 10.4.5.

10.2.6 Action

In the basic example of section 10.1, the action script is the script added to the monitoring system to kill all `yes` processes. The script runs when the condition is met that `CPUUser` crosses 50 jiffies/s.

An *action* is a standalone script or a built-in command that is executed when a condition is met, and has exit code 0 on success. The condition that is met can be:

- A `FAIL`, `PASS`, `UNKNOWN`, or `no data` from a health check
- A trigger condition. This can be a `FAIL` or `PASS` for conditional expressions.
- State flapping (section 10.2.9).

The actions that can be run are listed from within the `action` submode of the `monitoring` mode.

Example

```
[basecm11->monitoring->action]% list
Type        Name (key)       Run on   Action
----------- ---------------- ------------------------------------------------------
Drain       Drain            Active   Drain node from all WLM
```

```
Email       Send e-mail       Active   Send e-mail
Event       Event             Active   Send an event to users with connected client
ImageUpdate ImageUpdate       Active   Update the image on the node
PowerOff    PowerOff          Active   Power off a device
PowerOn     PowerOn           Active   Power on a device
PowerReset  PowerReset        Active   Power reset a device
Reboot      Reboot            Node     Reboot a node
Script      killallyesaction  Node     /cm/local/apps/cmd/scripts/actions/killallyes
Script      killprocess       Node     /cm/local/apps/cmd/scripts/actions/killprocess.pl
Script      remount           Node     /cm/local/apps/cmd/scripts/actions/remount
Script      testaction        Node     /cm/local/apps/cmd/scripts/actions/testaction
Shutdown    Shutdown          Node     Shutdown a node
Undrain     Undrain           Active   Undrain node from all WLM
```

The Base View equivalent is accessible via the navigation path:

`Monitoring` > `Actions` (figure 10.19, section 10.4.4)

Configuration of monitoring actions is discussed further in section 10.4.4.

10.2.7 Severity

Severity is a positive integer value that the administrator assigns for a trigger. It takes one of these 6 suggested values:

Value	Name in `cmsh`	Base View Icon	Description
0	`debug`		debug message
0	`info`		informational message
10	`notice`		normal, but significant, condition
20	`warning`		warning conditions
30	`error`		error conditions
40	`alert`		action must be taken immediately

Severity levels are used in the `AlertLevel` metric (section 10.2.8). They can also be set by the administrator in the return values of health check scripts (section 10.2.4).

By default the severity value is `15` for a health check `FAIL` response, `10` for a health check `UNKNOWN` response, and `0` for a health check `PASS` response (section 10.2.4).

10.2.8 AlertLevel

AlertLevel is a special metric. It is sampled and re-calculated when an event with an associated `Severity` (section 10.2.7) occurs. There are three types of `AlertLevel` metrics:

1. `AlertLevel (count)`: the *number* of events that are at `notice` level and `higher` . The aim of this metric is to alert the administrator to the *number* of issues.
2. `AlertLevel (max)`: simply the maximum severity of the latest value of all the events. The aim of this metric is to alert the administrator to the severity of the *most important* issue.
3. `AlertLevel (sum)`: the *sum* of the latest severity values of all the events. The aim of this metric is to alert the administrator to the *overall severity* of issues.

10.2.9 Flapping

Flapping, or *State Flapping*, is when a measurable trigger is detecting changes (section 10.4.5) that are too frequent. That is, the measurable goes in and out of the zone too many times over a number of samples.

In the basic example of section 10.1, if the CPUUser metric crossed the threshold zone 5 times within 5 minutes (the default values for flap detection), then it would by default be detected as flapping. A flapping alert would then be recorded in the event viewer, and a flapping action could also be launched if configured to do so.

10.2.10 Data Producer

A data producer produces measurables. Sometimes it can be a group of measurables, as in the measurables provided by a data producer that is being used:

Example

```
[basecm11->monitoring->measurable]% list -f name:25,producer:15 | grep ProcStat
BlockedProcesses          ProcStat
CPUGuest                  ProcStat
CPUIdle                   ProcStat
CPUIrq                    ProcStat
CPUNice                   ProcStat
CPUSoftIrq                ProcStat
CPUSteal                  ProcStat
CPUSystem                 ProcStat
CPUUser                   ProcStat
CPUWait                   ProcStat
CtxtSwitches              ProcStat
Forks                     ProcStat
Interrupts                ProcStat
RunningProcesses          ProcStat
```

Sometimes it may just be one measurable, as provided by a used data producer:

Example

```
[basecm11->monitoring->measurable]% list -f name:25,producer:15 | grep ssh2node
ssh2node                  ssh2node
```

It can even have no measurables, and just be an empty container for measurables that are not in use yet.

In cmsh all possible data producers (used and unused) can be listed as follows:

Example

```
[basecm11->monitoring->setup]% list
```

The equivalent in Base View is via the navigation path:

```
Monitoring > Data Producers
```

The data producers configured for an entity, such as a head node basecm11, can be listed with the monitoringproducers command:

Example

```
[basecm11->device[basecm11]]% monitoringproducers
                                                                                Node execution
Type           Name (key)                          ... Measurables              filters
-------------- ----------------------------------... ------------------------------------ --------------
```

```
AlertLevel     AlertLevel                        ... 3: AlertLevel:count, AlertLevel:maximum+ <1 in submode>
CMDaemonState  CMDaemonState                     ... 1: ManagedServicesOk                     <0 in submode>
CPU            CPUSampler                        ... 0:                                       <0 in submode>
Collection     NFS                               ... 44: nfs_server_file_anon, nfs_server_fi+ <1 in submode>
Collection     sample_sys_class_net              ... 51: sys_class_net_addr_assign_type:ens3+ <0 in submode>
Collection     smart                             ... 0:                                       <0 in submode>
DeviceState    DeviceState                       ... 1: DeviceStatus                          <1 in submode>
HealthCheck    chrootprocess                     ... 1: chrootprocess                         <1 in submode>
HealthCheck    cm-chroot-sw-img                  ... 1: cm-chroot-sw-img                      <1 in submode>
HealthCheck    cmsh                              ... 1: cmsh                                  <1 in submode>
HealthCheck    cuda-dcgm                         ... 1: cuda-dcgm                             <0 in submode>
HealthCheck    defaultgateway                    ... 1: defaultgateway                        <1 in submode>
HealthCheck    diskspace                         ... 1: diskspace                             <0 in submode>
HealthCheck    dmesg                             ... 1: dmesg                                 <0 in submode>
HealthCheck    exports                           ... 1: exports                               <1 in submode>
HealthCheck    interfaces                        ... 1: interfaces                            <0 in submode>
HealthCheck    ldap                              ... 1: ldap                                  <0 in submode>
HealthCheck    lustre                            ... 1: lustre                                <0 in submode>
HealthCheck    mounts                            ... 1: mounts                                <0 in submode>
HealthCheck    mysql                             ... 1: mysql                                 <1 in submode>
HealthCheck    ntp                               ... 1: ntp                                   <0 in submode>
HealthCheck    oomkiller                         ... 1: oomkiller                             <0 in submode>
HealthCheck    ssh2node                          ... 1: ssh2node                              <1 in submode>
JobMetadata    JobMetadataSampler                ... 4: job_metadata_allocated_cpu_cores, jo+ <1 in submode>
Metric         node_network_carrier_changes_total ... 1: node_network_carrier_changes_total   <0 in submode>
ProcMemInfo    ProcMemInfo                       ... 13: BufferMemory, CacheMemory, Hardware+ <0 in submode>
ProcMount      ProcMounts                        ... 10: FreeFiles:/, FreeFiles:/boot/efi, F+ <0 in submode>
ProcNetDev     ProcNetDev                        ... 27: BytesRecv:ens3, BytesRecv:eth0, Byt+ <0 in submode>
ProcNetSnmp    ProcNetSnmp                       ... 21: IpForwDatagrams, IpFragCreates, IpF+ <0 in submode>
ProcPidStat    ProcPidStat                       ... 7: MemoryUsed:cmd, RunTime:cmd, SystemT+ <0 in submode>
ProcStat       ProcStat                          ... 15: BlockedProcesses, CPUGuest, CPUIdle+ <0 in submode>
ProcVMStat     ProcVMStat                        ... 7: MajorPageFaults, OOM kill, PageFault+ <0 in submode>
SysBlockStat   SysBlockStat                      ... 20: IOInProgress:vda, IOInProgress:vdb,+ <0 in submode>
SysInfo        SysInfo                           ... 6: Cores, LoadFifteen, LoadFive, LoadOn+ <0 in submode>
UserCount      UserCount                         ... 3: TotalUser, TotalUserLogin, UniqueUse+ <1 in submode>
WlmSlot        WlmSlotSampler                    ... 0:                                       <1 in submode>
```

The displayed data producers are the ones configured for the entity, even if there are no measurables used by the entity.

Data producer configuration in Base View is discussed further in section 10.4.1.

Access Control For Monitoring Data

Access control to data producers: An access control setting for a data producer determines who can plot (via the measurables monitoring interface used by Base View and the User Portal), or view data (using the text-based interface of `cmsh` or `pythoncm`) from the measurables generated by a data producer. Thus, for example, the charts in the user portal (section 10.8) can be restricted according to the data producer that generates them.

There are three possible settings for access control for data producers. If the data producer is set to:

1. `Public`: then it means any user can, by default, plot/view data derived from that data producer. This is because by default a user has the token `PLOT_TOKEN` in their profile.

2. `Private`: then it means that a non-root user cannot, by default, plot/view data derived from that data producer. This is because, by default, non-root users do not have the token `PRIVATE_MONITORING_TOKEN` in their profile. If that token is in the profile, then the user has an elevated privilege, and can plot/view data, just like root.

3. `Individual`: then it means that a non-root user, by default, can plot/view the data only if the job associated with that data was run by that same non-root user. More verbosely, with the default user settings: the user who ran a job for which job measurables are produced by a data producer, must be the same as the user who wants to plot/view the data, or else the data cannot be plotted/viewed. The exception to this is, as already suggested, if the user that wants to plot/view the data is root, or has a user profile with the token `PRIVATE_MONITORING_TOKEN`. In that case the data can be plotted/viewed.

   NOTE: In more detail, individual access control is meant for job-based measurables, and works like this:

   All monitoring data is stored per (entity, measurable) pair.

   If the measurable has an access control value of `individual`, then a user check is performed. If the login name is the same as the user that owns the entity, then data can be plotted/viewed. If the user check does not match, then no data is returned.

   Jobs—which are entities—are owned by the user that ran the job. Similarly Prometheus (entities) can have a `user="Alice"` label set, to define ownership. No other entity managed by BCM is owned by a user.

   For all unowned entities, individual access is equivalent to private access.

On a regular BCM cluster, only a few low level data producers are set to `private`. An administrator can decide to set a data producer access control value to one of the three possible values, by using the `setup` submode of `monitoring` mode:

Example

```
[root@basecm11 ~]# cmsh
[basecm11]% monitoring setup
[basecm11->monitoring->setup]% use mounts
[basecm11->...[mounts]]% set access private
[basecm11->...*[mounts*]]% commit
```

If a data producer is newly added, then by default its access control value is set to `public`. Changing this to `private` at a later time means that access to past and future data values from that data producer are affected by the `private` setting. If access is changed once more back to `public`, then it means that access to past and future data values are once again viewable and plottable by all users.

The current settings for access control for the data producers can be seen with:

Example

```
[root@basecm11 ~]# cmsh -c "monitoring setup; list -f name:40,access"
name (key)                               access
---------------------------------------- --------------------
AggregateNode                            Public
AlertLevel                               Public
BigDataTools                             Public
CMDaemonState                            Public
Cassandra                                Public
ClusterTotal                             Public
...
```

Access control to measurables: Measurables can also have access controls.

Access control for measurables is by default inherited from the data producer that generates it. It can be overwritten:

Example

```
[root@basecm11 ~]# cmsh
[basecm11]% monitoring measurable
[basecm11->monitoring->measurable]% use loadone
[basecm11->...[LoadOne]]% get access
Public (SysInfo)
[basecm11->...[LoadOne]]% set access private
[basecm11->...*[LoadOne*]]% commit
```

A measurable can thus take an access control value of `Public`, `Private`, or `Individual`. It can also explicitly be set to a value of `inherit`, which sets it to the value of its data producer. The inheritance is indicated in `cmsh` by enclosing the parent data producer in parentheses, as shown in the preceding example.

The current settings for access control for the measurables can be seen with:

Example

```
[root@basecm11 ~]# cmsh -c "monitoring measurable; list -f name:40,access"
name (key)                               access
---------------------------------------- --------------------
AlertLevel                               Public
AlertLevel                               Public
AlertLevel                               Public
AvgJobDuration                           Public
BlockedProcesses                         Public
...
```

10.2.11 Conceptual Overview: The Main Monitoring Interfaces Of Base View

Base View, besides having the default settings mode, has some other display modes and logging view modes that can be selected via the 8 icons in the top right corner of the Base View standard display (figure 10.5):

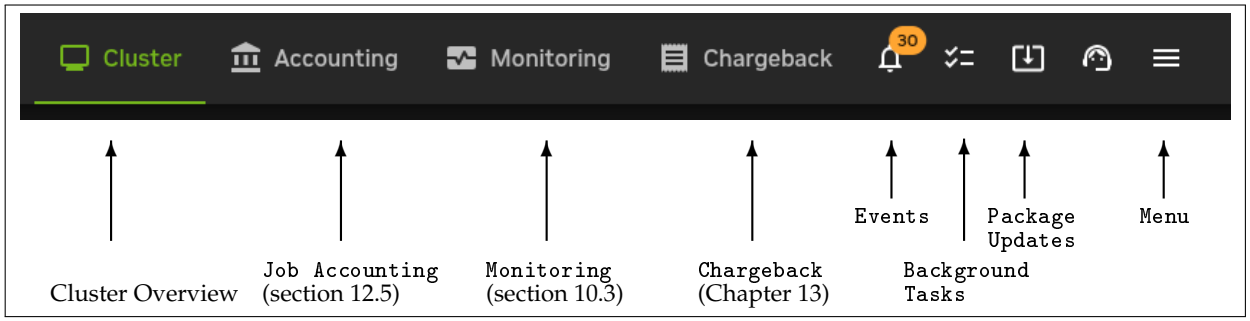


Figure 10.5: Base View: Top Right Corner Icons

The 8 icons are described from left to right next:

1. `Cluster Overview` icon has the browser display the main cluster overview page (figure 2.3).
2. The `Accounting` icon brings up a window that allows visualization of job resources used by users, although it can be used to visualize job resources used by other entities that are used as a classifier. Job accounting (Chapter 12) is helpful in tracking resources consumed by users.
3. The `Monitoring` mode allows visualization of the data values collected according to the specifications of the Base View `Monitoring` resource. The visualization allows graphs to be configured, and is discussed further in section 10.3.

4. The `Chargeback` mode allows the monitoring of resources requested over a period for jobs run by selected groups (Chapter 13).
5. The `Events` icon allows logs of events (section 10.10) to be viewed.
6. The `Background tasks` icon allows background tasks to be viewed.
7. The `Package Updates` icon lists packages that can be updated. This is the Base View equivalent of the `packages --outdated` command in `cmsh` (section 9.1.1).
8. The ≡ icon (hamburger icon, collapsed menu icon) expands to show, among other options, a dark mode toggle, and a logout option (figure 10.6):

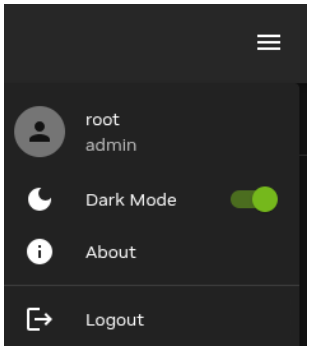


Figure 10.6: Base View Top Right Hand Corner Menu items

The monitoring aspects of the second and third icons of figure 10.5 are discussed in greater detail in the sections indicated.

10.3 Monitoring Visualization With Base View

The `Monitoring` icon in the menu bar of Base View (item 3 in figure 10.5) launches an intuitive visualization tool that is the main GUI tool for getting a feel of the system's behavior over periods of time. With this tool the measurements and states of the system can be viewed as resizable and overlayable graphs. The graphs can be zoomed in and out on over a particular time period, the graphs can be laid out on top of each other or the graphs can be laid out as a giant grid. The graph scale settings can also be adjusted, stored and recalled for use the next time a session is started.

An alternative to Base View's visualization tool is the command-line `cmsh`. This has the same functionality in the sense that data values can be selected and studied according to configurable parameters with it (section 10.6). The data values can even be plotted and displayed on graphs with `cmsh` with the help of unix pipes and graphing utilities. However, the strengths of monitoring with `cmsh` lie elsewhere: `cmsh` is more useful for scripting or for examining pre-decided metrics and health checks rather than a quick visual check over the system. This is because `cmsh` needs more familiarity with options, and is designed for text output instead of interactive graphs. Monitoring with `cmsh` is discussed in sections 10.5 and 10.6.

Visualization of monitoring graphs with Base View is now described.

10.3.1 The Monitoring Window

If the `Monitoring` icon is clicked on from the menu bar of Base View (figure 10.5), then a monitoring window for visualizing data opens up. By default, this displays a dashboard called `Critical Services` which has two plot panels.The plot panels have graph axes with a time scale going back some time on the x-axis, and with a y-axis that allows measurable data values plotted against time (figure 10.7).

By default, the data values that are plotted are for the `ManagedServicesOK` health check. The first panel displays the plots for the head node, the second panel displays the plots for all the regular nodes in the `default` category.

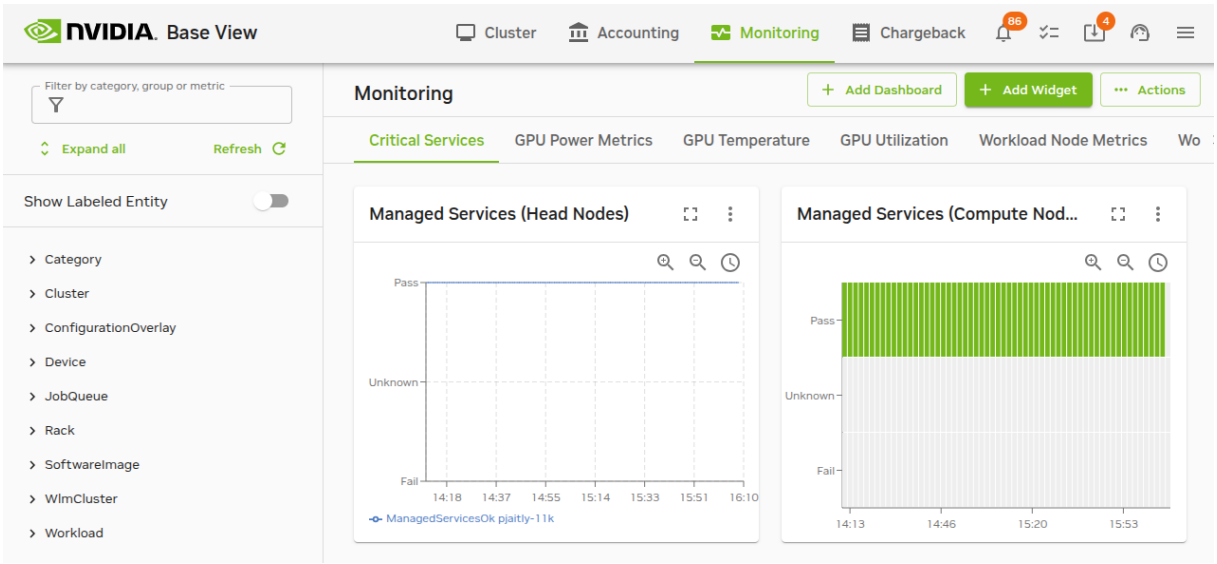


Figure 10.7: Base View Monitoring Window: Default Plot Panels

Finding And Selecting The Measurable To Be Plotted

To plot measurables, the entity which it belongs to should be selected from the navigation menu on the left-hand side. Once that has been selected, a class for that measurable can be chosen, and then the measurable itself can be selected. For example, to plot the measurable `CPUUser` for a head node `basecm11`, it can be selected from the navigation path:

Monitoring icon > `Device > head node > CPU > CPUUser`

Sometimes, finding a measurable is easier if the `Expand all` widget is used, together with the filter box. Typing in `CPUUser` in the filter box then shows all the measurables with that text (figure 10.8). The search is case-insensitive.

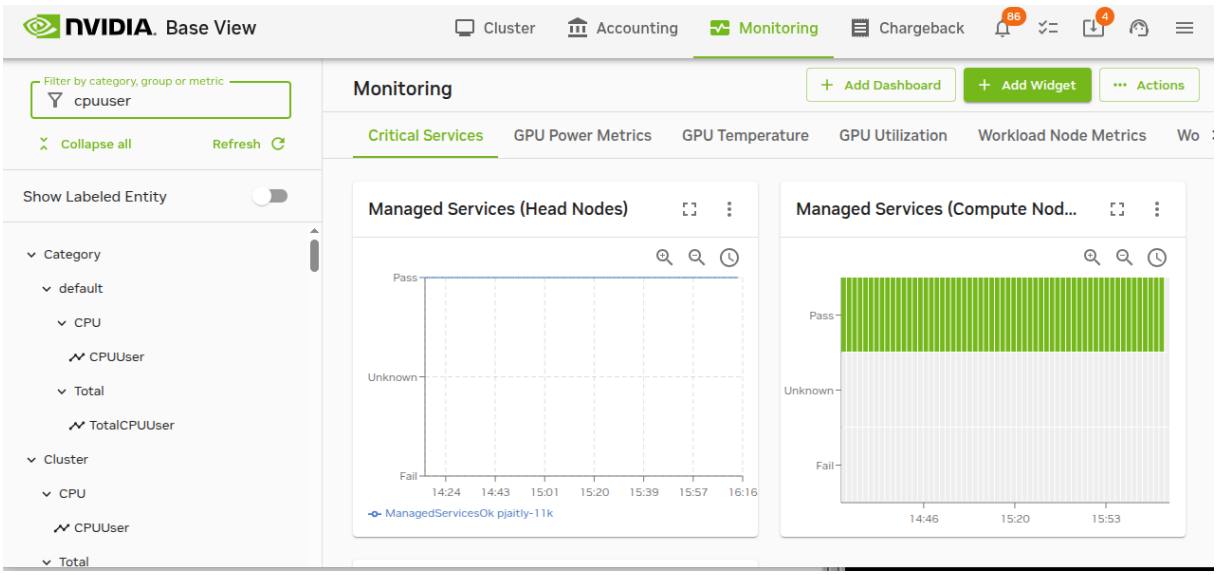


Figure 10.8: Base View Monitoring Window: Search Box In Navigation

The search box can handle some simple regexes too, with `.*` and `|` taking their usual meaning:

Example

- `node001.*cpuuser`: select a measurable with a data path that starts with `node001` and ends with `cpuuser`, with 0 or more characters of any kind in between.
- `(node001|node002).*cpuuser`: as for preceding example, but including `node002` as an alternative to `node001`.

The / (forward slash) allows filtering according to the data path. It corresponds to the navigation depth in the tree hierarchy:

Example

- `node001/cpu/cpuuser`: search for a measurable with a data path that matches `node001/cpu/cpuuser`

Plotting The Measurable

Once the measurable is selected, it can be drag-and-dropped into a plot panel. This causes the data values to be plotted.

When a measurable is plotted into a panel, two graph plots are displayed. The smaller, bottom plot, represents the polled value as a bar chart. The larger, upper plot, represents an interpolated line graph. Different kinds of interpolations can be set. To get a quick idea of the effect of different kinds of interpolations, `https://bl.ocks.org/mbostock/4342190` is an interactive overview that shows how they work on a small set of values.

The time axes can be expanded or shrunk using the mouse wheel in the graphing area of the plot panel. The resizing is carried out centered around the position of the mouse pointer.

10.4 Monitoring Configuration With Base View

This section is about the configuration of monitoring for measurables, and about setting up trigger actions.

If Base View is running in the standard `Cluster Overview` modeselecting `Monitoring` from the navigation menu resources makes the following menu items available for managing or viewing:

- `Data Producers` (section 10.4.1)
- `Measurables` (section 10.4.2)
- `Consolidators` (section 10.4.3)
- `Actions` (section 10.4.4)
- `Triggers` (section 10.4.5)
- `Health status` (section 10.4.6)
- `All Health checks` (section 10.4.7)
- `Standalone Monitored Entities` (section 10.4.8)
- `Queries` (section 10.4.9)

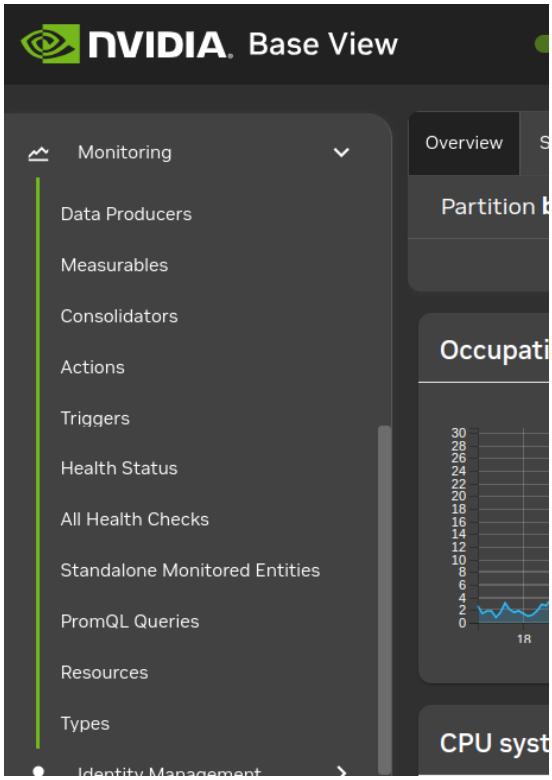


Figure 10.9: Base View Monitoring Configuration Settings

These settings (figure 10.9) are now discussed in detail.

10.4.1 Monitoring Configuration: Data Producers

The navigation path:
`Monitoring` > `Dataproducers`
opens up the `Monitoring Data Producer list` screen, which lists all the data producers (figure 10.10).

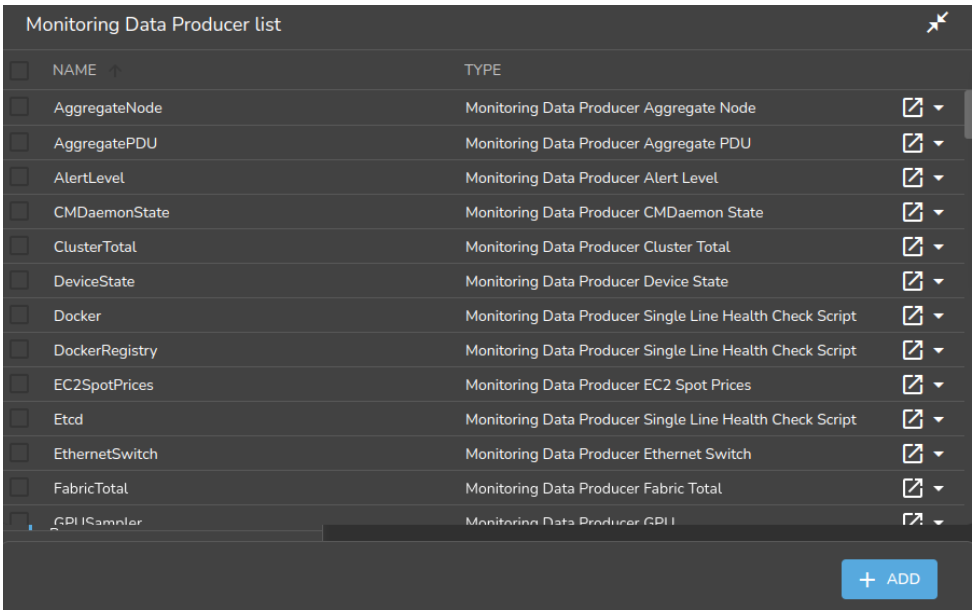


Figure 10.10: Base View Monitoring Configuration Data Producers

Data producers are introduced in section 10.2.10.

Each data producer can have its settings edited within a subwindow. For example, the `ProcStat` data producer, which produces data for several measurables, including `CPUUser`, has the settings shown in figure 10.11:

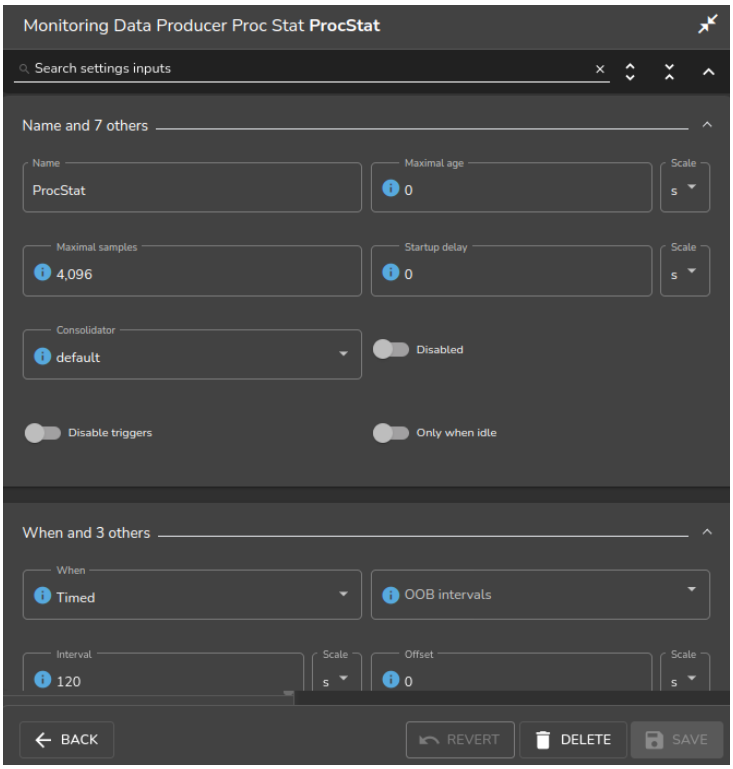


Figure 10.11: Base View Monitoring Configuration Data Producer: `ProcStat`

When the data producer takes samples to produce data, run length encoding (RLE) is used to compress the number of samples that are stored as data. Consolidation is carried out on the RLE samples. Consolidation in BCM means gathering several data values, and making one value from them over time periods. Consolidation is done as data values are gathered. The point at which data values are discarded, if ever, is thus not dependent on consolidation.

The data producer settings that are seen in the subwindow of figure 10.11 include the following:

- `Maximal samples`: the maximum number of RLE samples that are kept. If set to `0`, then the number of samples is not considered.
- `Maximal Age`: the maximum age of RLE samples that are kept. If `Maximal Age` is set to `0` then the sample age is not considered.

  With `Maximal samples` and `Maximal Age`, the first of the rules that is reached is the one that causes the exceeding RLE samples to be dropped.

  Samples are kept forever if `Maximal samples` and `Maximal Age` are both set to `0`. This is discouraged due to the risk of exceeding the available data storage space.

  If the number of samples is reduced, then previously consolidated data values are automatically discarded after about 30 minutes. Changing parameters should therefore be done with some care.
- `Interval`: the interval between sampling, in seconds.
- `Offset`: A time offset from start of sampling. Some sampling depends on other sampling to be carried out first. This is used, for example, by data producers that rely on sampling from other

data producers. For example, the AggregateNode data producer, which has measurables such as TotalCPUIdle and TotalMemoryFree. The samples for AggregateNode depend upon the ProcStat data producer, which produces the CPUIdle measurable; and the ProcMemInfo data producer, which produces the MemoryFree measurable.

- Fuzzy offset: a multiplier in the range from 0 to 1. It is multiplied against the sampling time interval to fix a maximum value for the time offset for when the sampling takes place. The actual offset used per node is spread out reasonably evenly within the range up to that maximum time offset.

  For example, for a sampling time interval of 120s:

  If the offset is 0, then there is no offset, and the sampling is attempted for all nodes at time instant when the interval restarts. This can lead to an overload at the time of sampling.

  If, on the other hand, the offset is 0.25, then the sampling is done within a range offset from the time of sampling by a maximum of $0.25 \times 120\text{s} = 30\text{s}$. So, each node is sampled at a time that is offset by up to 30s from when the 120s interval restarts. From the time the change in value of the fuzzy offset starts working, the offset is set for each node. The instant at which sampling is carried out on a node then differs from the other nodes, even though each node still has an interval of 120s between sampling. An algorithm is used that tends to even out the spread of the instants at which sampling is carried out within the 30s range. The spreading of sampling has the effect of reducing the chance of overload at the time of sampling.

- Consolidator: By default, set to the default group. The default group consolidates (summarizes) the RLE samples over periods of an hour. Consolidators are explained further in section 10.4.3.

- Node execution filters (page 617): A way to filter execution (restrict execution) of the data producer. It tells BCM where the data producer runs. If not set, then the data producer runs on all nodes managed by CMDaemon.

- Execution multiplexer (page 621): A way to multiplex execution (have execution work elsewhere) for a data producer. The object tells BCM about the entities that the data producer is executing sampling at. Thus a data producer gathers data at the nodes defined by the node execution filter, and the data producer executes sampling at other entities, which are defined into groups called execution multiplexers.

- When: This has four possible values:

  1. Timed: Data producer is run at a periodic Interval. This is the default.
  2. On demand: Data producer is only run on demand, and not at a periodic Interval.
  3. Out of band: Data producer is only run on out of band connections.
  4. On start: Data producer is only run on start and not at a periodic Interval.

- Only when idle: By default a data producer runs regardless of how busy the nodes are. However, if the Only when idle setting is enabled, then the data producer runs only when the node is idle. Idle is a condition that is defined by the metric condition LoadOne>1 (page 994).

10.4.2 Monitoring Configuration: Measurables

The Base View navigation path Monitoring > Measurables lists the available measurables (figure 10.12):

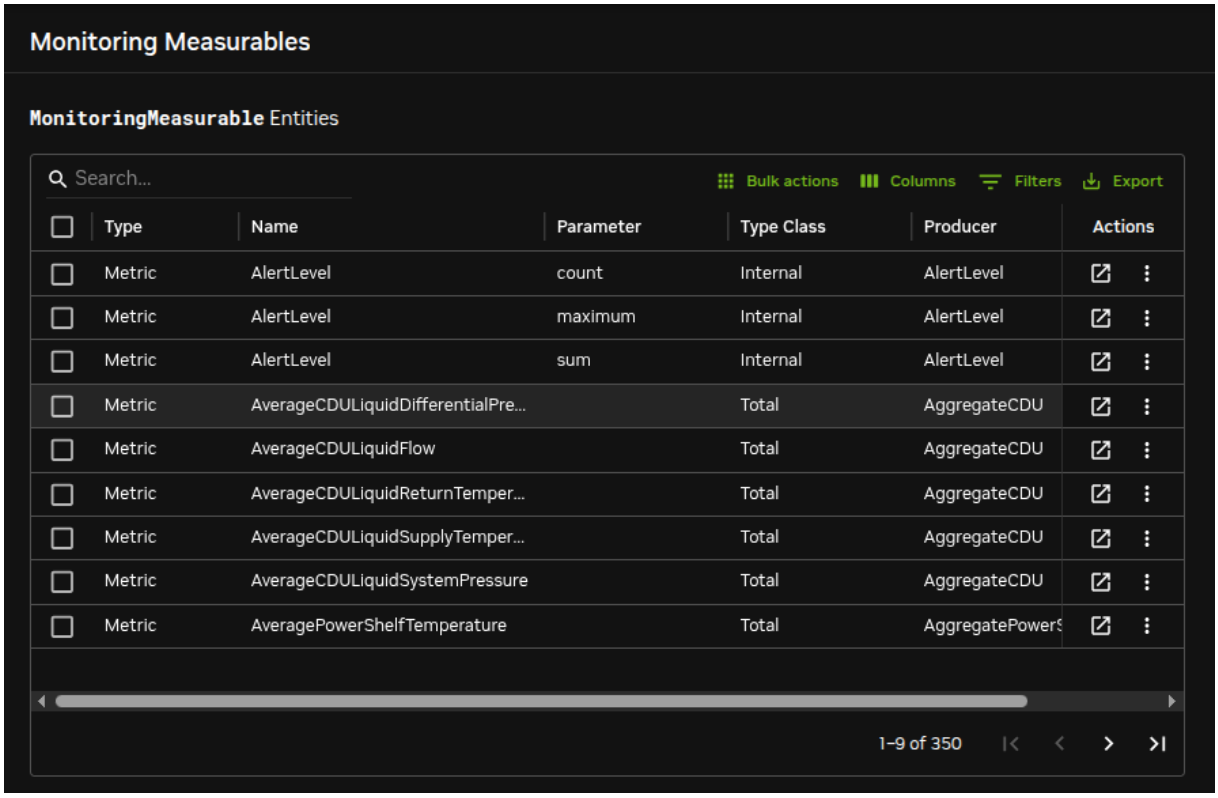


Figure 10.12: Base View Monitoring Configuration Measurables

There are many measurables, so using the search box can be handy.

The icon opens a window for settings to a particular measurable. In figure 10.13 metric settings are seen for the `AlertLevel` metric:

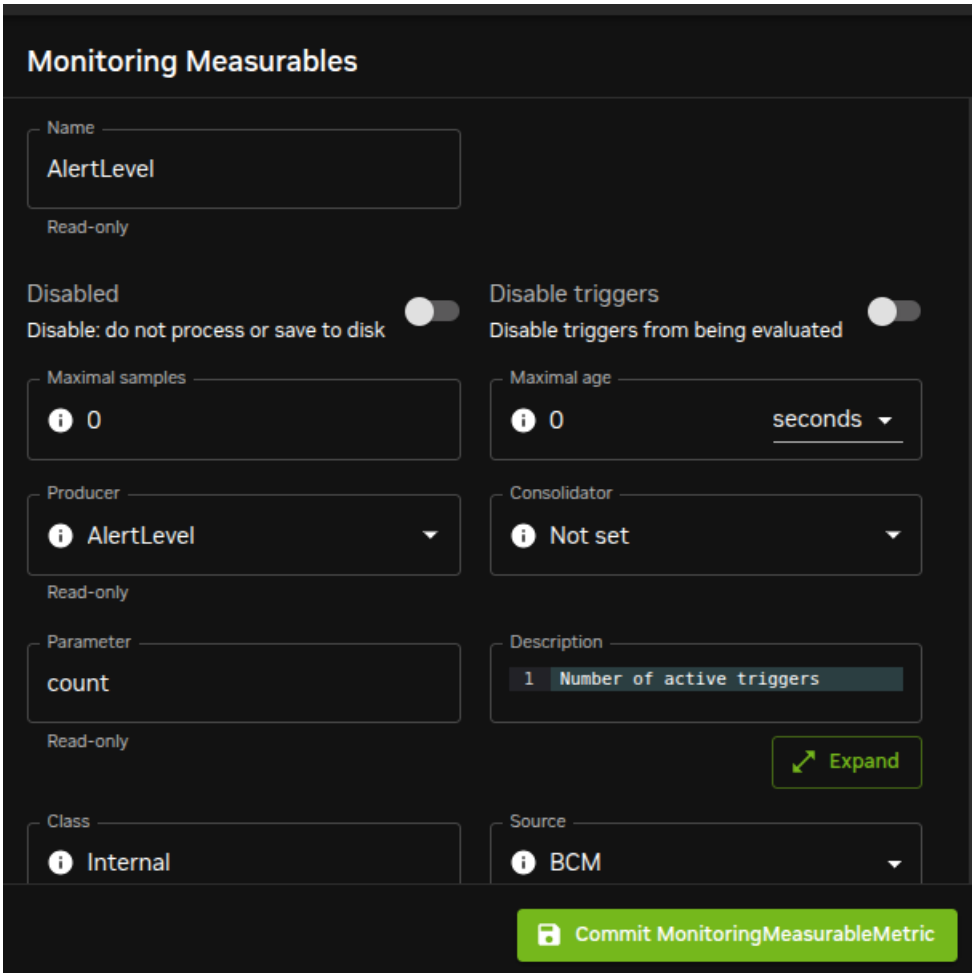


Figure 10.13: Base View Monitoring Configuration Measurables Settings Window

The sampling options include: `Maximal age`, `Maximal samples`, and `Consolidator`. The sampling options work as described for data producers (section 10.4.1).

Other options for a metric are setting the `Maximum` and `Minimum` values, the `Unit` used, and whether the metric is `Cumulative`.

If a metric is cumulative, then it is monotonic. Monotonic means that the metric only increments (is cumulative), as time passes. In other words, if the metric is plotted as a graph against time, with time on the x-axis, then the metric never descends. Normally the increments are from the time of boot onward, and the metric resets at boot. For example, the number of bytes received at an interface is cumulative, and resets at boot time.

Usually the cluster administrator is only interested in the differential value of the metric per sample interval. That is, the change in the value of the current sample, from its value in the preceding sample. For example, bytes/second, rather than total number of bytes up to that time from boot.

10.4.3 Monitoring Configuration: Consolidators

Introduction To Consolidators

The concept of consolidators is explained using simple ascii graphics in Appendix K, while the `cmsh` interface to the `consolidators` submode is discussed in section 10.5.2.

In this current section, the Base View interface to consolidators is discussed.

In Base View, the navigation path `Monitoring > Consolidator` lists all consolidator groups (figure 10.14). There are two pre-existing consolidator groups: `default` and `none`.

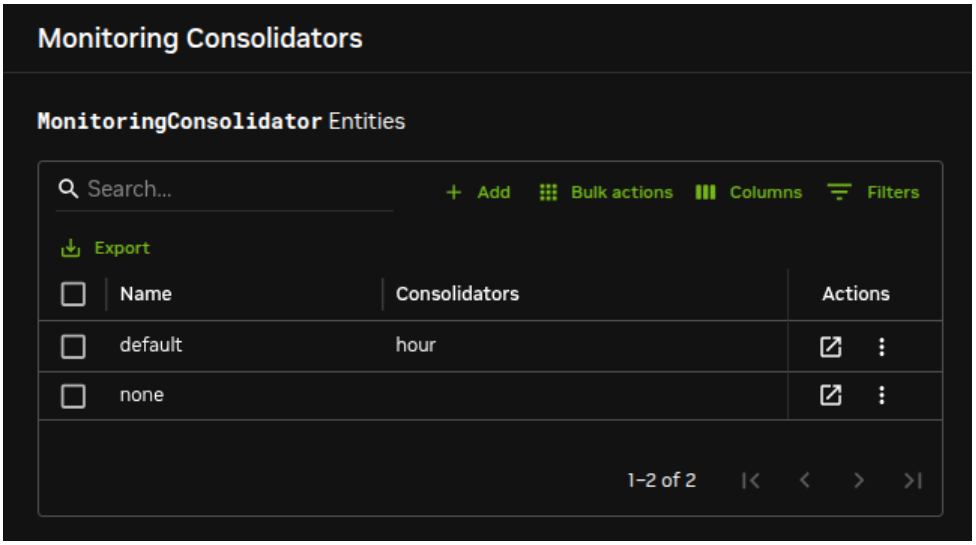


Figure 10.14: Base View Monitoring Consolidator list

Subwindows allow the consolidator components (consolidator items) to be created or modified (figure 10.15).

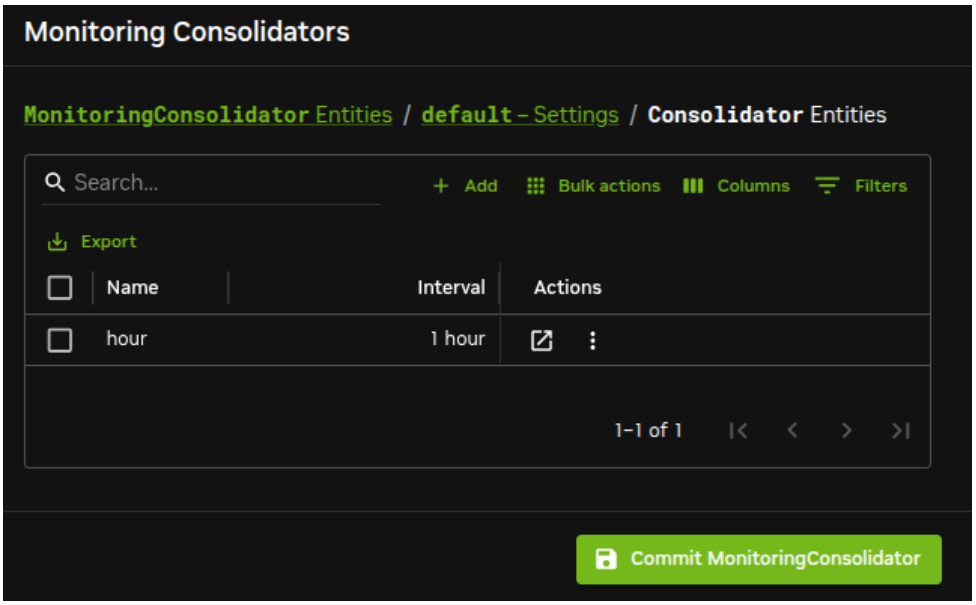


Figure 10.15: Base View Monitoring Configuration Consolidator Items

The `none` Consolidator Group

The `none` consolidator group has no consolidators. Using a consolidator group of `none` for a measurable or data producer means that samples are not consolidated. This can be dangerous if the cluster is more likely to run out of space due to unrestrained sampling. Unrestrained sampling can occur, for example, if `Maximal Age` and `Maximal samples` (section 10.4.1) for data producers are both set to `0`.

The `default` Consolidator Group

In BCM 11, the `default` consolidator group consists of the consolidator `hour` only. If the cluster was upgraded to BCM 11 from BCM 10, then legacy `day` and `week` consolidators are also present. The consolidators are, unsurprisingly, defined to consolidate the samples in intervals of an hour, day, or week.

A consolidated value is generated on-the-fly. So, for example, during the hour that samples of a

measurable come in, the `hour` consolidator uses the samples to readjust the consolidated value for that hour. When the hour is over, the consolidated value is stored for that hour as the data value for that hour, and a new consolidation for the next hour begins.

Consolidator values are kept, as for sample values, until the `Maximal Age` and `Maximal sample` settings prevent data values being kept.

In BCM 11 and later, the maximal samples for hours is set by default to 2^{16} hours, which is about 7.5 years.

Other Consolidator Group Possibilities

Other sets of custom intervals can also be defined. For example, instead of the `default` consolidator group, which could consist of consolidators of an `hour`, a `day`, and a `week`; a similar group called the `decimalminutes` consolidator group can be added (figure 10.16), consisting of consolidators of `1min`, `10min`, `100min`, `1000min`, `10000min`:

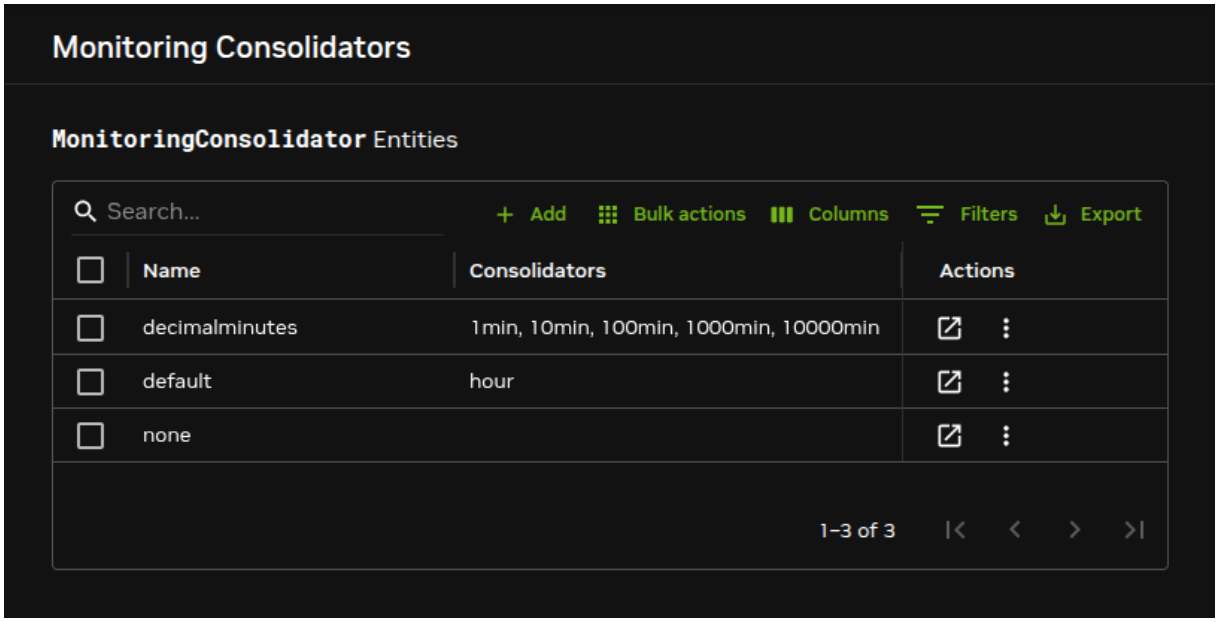


Figure 10.16: Base View Monitoring Configuration Consolidators: `decimalminutes` Consolidator Group

Each consolidator can be created and given appropriate definitions for the intervals and other properties (figure 10.17):

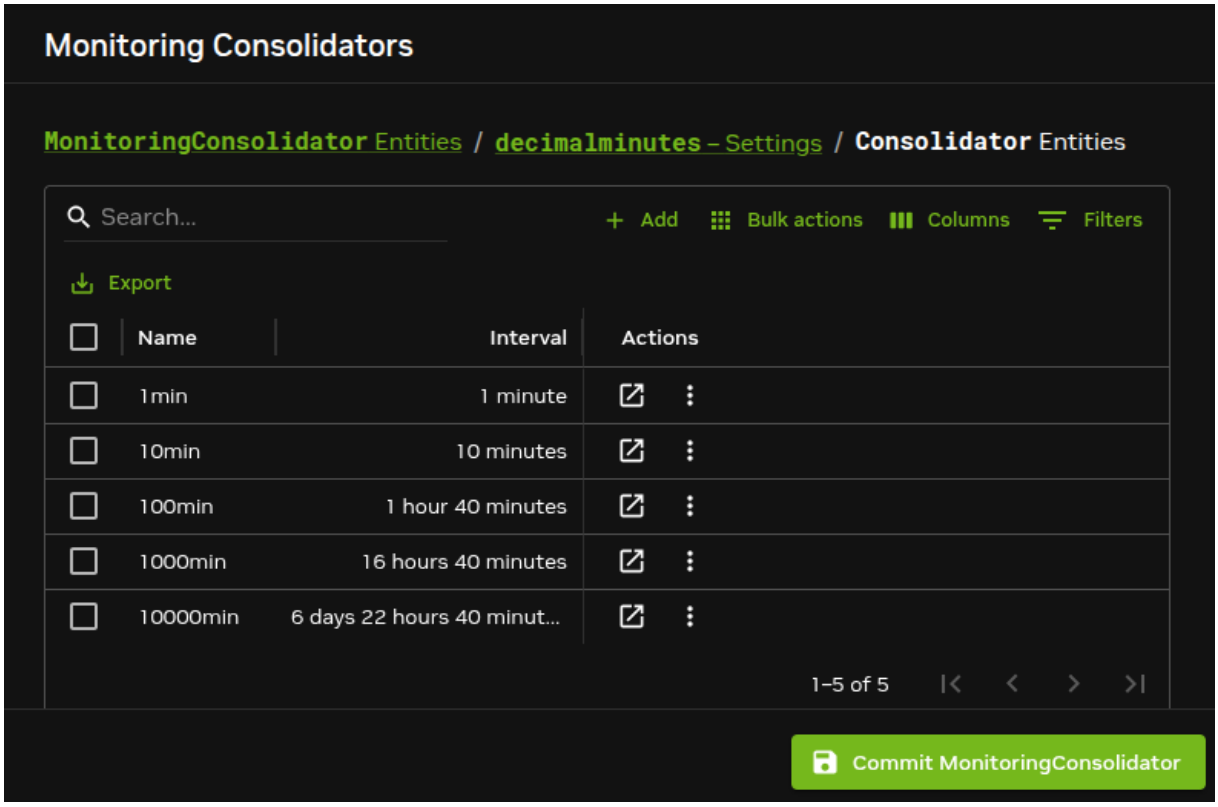


Figure 10.17: Base View Monitoring Configuration Consolidators: `decimalminutes` Consolidator Definitions

For the record, values for the `decimalminutes` consolidators that make sense could be as follows:

Name	Maximal Age/minutes	Maximal Samples	Interval/minutes
1min	10	65536	1
10min	100	65536	10
100min	1000	65536	100
1000min	10000	65536	1000
10000min	0	65536	10000

The values in the table are examples of settings of consolidator items. The settings of consolidator items are discussed next.

Consolidator Item Settings

Consolidator items (consolidator components) are the component members of the consolidator groups. The items have settings as properties, which can be managed (figure 10.18).

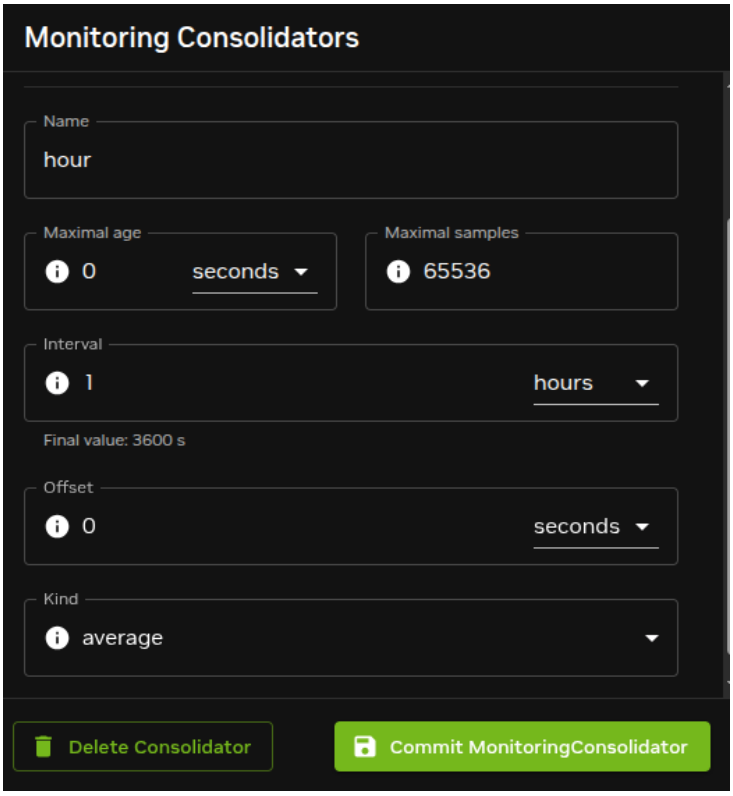


Figure 10.18: Base View Monitoring Configuration Consolidators: Consolidator Item Settings

The consolidator item `hour`, which is within the `default` consolidators group, can have its properties edited using the navigation path:

`Monitoring > Consolidators[default]` > `Consolidators` > `hour`

The properties that can be set for a consolidator item are:

- `Name`: The name of the consolidator item. By default, for the consolidator group `default`, the consolidator item `hour` is already set up, with an interval of 1 hour, and no limit on the maximum number of samples.
- `Maximal samples`: The maximum number of samples that are stored for that consolidator item. This should not be confused with the `Maximal samples` of the measurable being consolidated.
- `Interval`: The time period (in seconds) covered by the consolidator sample. For example, the consolidator with the name `hour` has a value of 1 hour (3600s). The property should not be confused with the time period between samples of the measurable being consolidated.
- `Offset`: The time offset from the default consolidation time, explained in more detail shortly.
- `Kind`: The kind of consolidation that is done on the raw data samples. The value of `kind` is set to `average` by default. The output result for a processed set of raw data—the consolidated data point—is an average, a maximum or a minimum of the input raw data values. `Kind` can thus have the value `average`, `maximum`, or `minimum`. The value of `kind` is set to `average` by default.

  For a given consolidator, when one `Kind` is changed to another, the historically processed data values become inconsistent with the newer data values being consolidated. Previous consolidated data values for that consolidator are therefore discarded during such a change.

To understand what Offset means, the Maximal samples of the measurable being consolidated can be considered. This is the maximum number of raw data points that the measurable stores. When this maximum is reached, the oldest data point is removed from the measurable data when a new data point is added. Each removed data point is gathered and used for data consolidation purposes.

For a measurable that adds a new data point every Interval seconds, the time $t_{\text{raw gone}}$, which is how many seconds into the past the raw data point is removed, is given by:

$$t_{\text{raw gone}} = (\text{Maximal samples})_{\text{measurable}} \times (\text{Interval})_{\text{measurable}}$$

This value is also the default consolidation time, because the consolidated data values are normally presented from $t_{\text{raw gone}}$ seconds ago, to further into the past. The default consolidation time occurs when the Offset has its default, zero value.

If however the Offset period is non-zero, then the consolidation time is offset, because the time into the past from which consolidation is presented to the user, $t_{\text{consolidation}}$, is then given by:

$$t_{\text{consolidation}} = t_{\text{raw gone}} + \text{Offset}$$

The monitoring visualization graphs then show consolidated data from $t_{\text{consolidation}}$ seconds into the past, to further into the past.

NOTE: For completeness: the time $t_{\text{consolidation gone}}$, which is how many seconds into the past the consolidated data goes and is viewable, is given by an analogous equation to that of the equation defining $t_{\text{raw gone}}$:

$$t_{\text{consolidation gone}} = (\text{Maximalsamples})_{\text{consolidation}} \times (\text{Interval})_{\text{consolidation}}$$

10.4.4 Monitoring Configuration: Actions

Actions are introduced in section 10.2.6, and are available in Base View with the navigation path Monitoring > Actions. The Actions window (figure 10.19) displays actions that BCM provides by default, and also displays any custom actions that have been created:

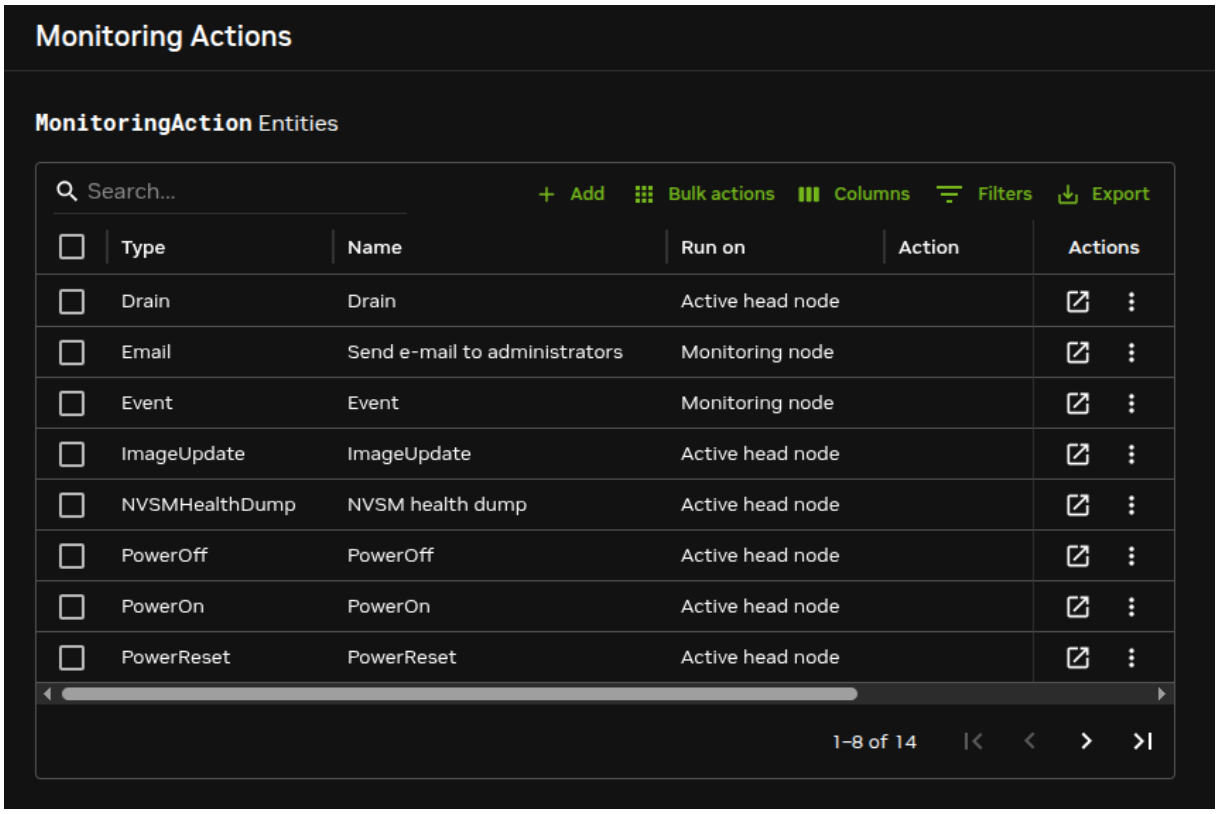


Figure 10.19: Base View Monitoring Configuration: Actions

The custom killallyes script from the basic example of section 10.1 would show up here if it has

been implemented.

Actions are triggered, by triggers (section 10.4.5).

By default, the following actions exist:

- `PowerOn`: Powers on the node
- `PowerOff`: Powers off the node
- `PowerReset`: Hard resets the node
- `Drain`: Drains the node (does not allow new jobs on that node)
- `Undrain`: Undrains the node (allows new jobs on that node)
- `Reboot`: Reboots node via the operating system
- `Shutdown`: Shuts the node down via the operating system
- `ImageUpdate`: Updates the node from the software image
- `NVSMHealthDump`: NVSM (section 10.13) health dump (for DGX hardware only)
- `Event`: Sends an event to users connected with `cmsh` or Base View
- `killprocess`: A script to kill a process
- `remount`: A script to remount all devices
- `testaction`: A test script
- `Send e-mail to administrators`: Sends an e-mail out

The preceding actions show their options when the associated  icon is clicked. A new window with the options opens up. The following options are among those then displayed:

- `Run on`: What nodes the action should run on. Choices are:
  - `Active head node`: the action runs on the active node only
  - `Node`: the action runs on the triggering node
  - `Monitoring node`: the action runs on the monitoring node
- `Allowed time`: The time interval in the 24 hour clock cycle that the action is allowed to start running. The interval can be restricted further to run within certain days of the week, months of the year, or dates of the month. Days and months must be specified in lower case.

  The Base View interface for setting the time can be used to set the allowed times.

  The allowed times have a format that is also used in `cmsh`. Rather than defining a formal syntax, some `cmsh` examples are given of possible formats, with explanations:

  - `november-march`: November to March. The months April to October are forbidden.
  - `november-march{monday-saturday}`: As in the preceding, but all Sundays are also forbidden.
  - `november-march{monday-saturday{13:00-17:00}}`: Restricted to the period defined in the preceding example, and with the additional restriction that the action can start running only during the time 13:00-17:00.
  - `09:00-17:00`: All year long, but during 09:00-17:00 only.
  - `monday-friday{9:00-17:00}`: All year long, but during 9:00-17:00 only, and not on Saturdays or Sundays.

- november-march{monday-saturday{13:00-17:00}}: Not in April to October. In the other months, only on Mondays to Saturdays, from 13:00-17:00.
- may-september{monday-friday{09:00-18:00};saturday-sunday{13:00-17:00}}: May to September, with: Monday to Friday 09:00-18:00, and Saturday to Sunday 13:00-17:00.
- may{1-31}: All of May.
- may,september{1-15}: All of May, and only September 1-15.
- may,september{1-15{monday-friday}}: All of May. And only September 1-15 Monday to Friday.

An EBNF grammar for allowed times is given in section 3.2.1 of the *Developer Manual*.

The following action scripts have some additional options:

- Send e-mail to administrators: Additional options here are:
  - Info: body of text inserted into the default e-mail message text. This is the text that replaces the {infomessage} placeholders in the email template formats seen in the file /cm/local/apps/cmd/scripts/actions/sendemail.py
  - Recipients: a list of recipients
  - All administrators: uses the list of users in the Administrator e-mail setting in partition[base] mode
- killprocess, and testaction: Additional options for these are:
  - Arguments: text that can be used by the script.
  - Script: The location of the script on the file system.

10.4.5 Monitoring Configuration: Triggers

Triggers are introduced in section 10.2.5. The Triggers window (figure 10.20) allows actions (section 10.2.6) to be triggered based on conditions defined by the cluster administrator.

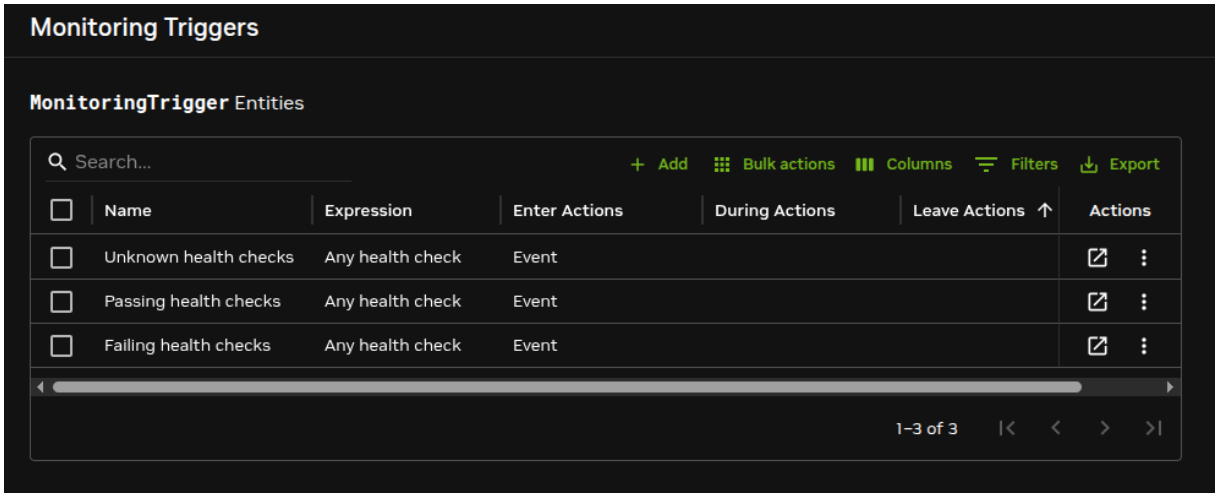


Figure 10.20: Base View Monitoring Configuration: Triggers

Change Detection For Triggers

Triggers launch actions by detecting changes in the data of configured measurables. The detection of these changes can happen:

- When a threshold is crossed. That is:

  - either the preceding sample value was not in the zone, and the sample value now has entered the zone,
  - or the preceding sample value was in the zone, and the sample value now has left the zone

- When the zone remains crossed. That is: the sample value now as well as the preceding sample value are both within the zone of a crossed threshold.
- When state flapping is detected. This is when the threshold is crossed repeatedly (5 times by default) within a certain time period (5 minutes by default).

The monitoring configuration dialog triggers have four possible action launch configuration options to cover these cases:

1. `Enter actions`: if the sample has entered into the zone and the previous sample was not in the zone. This is a threshold-crossing change.
2. `Leave actions`: if the sample has left the zone and the previous sample was in the zone. This is also a threshold-crossing change.
3. `During actions`: if the sample is in the zone, and the previous sample was also in the zone.
4. `State flapping actions`: if the sample is entering and leaving the zone within a particular period (`State flapping period`, 5 minutes by default) a set number of times (`State flapping count`, 5 by default).

Pre-defined Triggers: Passing, Failing, And Unknown Health Checks

By default, the only triggers that are pre-defined are the following three health check triggers, which use the `Enter actions` launch configuration option, and which have the following default behavior:

- `Failing health checks`: If a health check fails, then on entering the state of the health check failing, an event is triggered as the action, and a severity of `15` is set for that health check.
- `Passing health checks`: If a health check passes, then on entering the state of the health check passing, an event is triggered as the action, and a severity of `0` is set for that health check.
- `Unknown health checks`: If a health check has an unknown response, then on entering the state of the health check returning an unknown response, an event is triggered as the action, and a severity of `10` is set for that health check.

Example: carry out a triggered action: `cmsh` or Base View can be used for the configuration of carrying out an e-mail alert action (that is: sending out an e-mail) that is triggered by failing health checks.

- A `cmsh` way to configure it is:

  The e-mail action (`send\ e-mail\ to\ administrators`) is first configured so that the right recipients get a useful e-mail.

```
[root@basecm11 ~]# cmsh
[basecm11]% monitoring action
[basecm11->monitoring->action]% use send\ e-mail\ to\ administrators
[basecm11->...[send e-mail to administrators]]% append recipients user1@example.com
[basecm11->...*[send e-mail to administrators*]]% commit
```

  Here, email alerts would go to `user1@example.com`, as well as to anyone already configured in `administratore-mail`. Additional text can be set in the body of the e-mail by setting a value for `info`.

  The trigger can be configured to run the action when the health check enters a state where its value is `true`:

```
[basecm11->monitoring->action[use send e-mail to administrators]]% monitoring trigger
[basecm11->monitoring->trigger]% use failing\ health\ checks
[basecm11->monitoring->trigger[Failing health checks]]% append enteractions send\ e-mail\ to\ administrators
[basecm11->monitoring->trigger*[Failing health checks*]]% commit
```

The settings can be viewed with the `show` command. TAB-completion prompting can be used to suggest possible values for the settings.

- A Base View way to carry out the configuration is using the navigation path:

  `Monitoring > Actions > Send e-mail to administrators`

  This can be used to set the recipients and other items, and the configuration can then be saved.

  The email action can then be configured in Base View using the navigation path:

  `Monitoring > Triggers > Failing Health Checks >` `> Enter Actions > Send E-mail to Administrators`

  The `Send E-mail to Administrators` action should be added to the items in the `Enter actions` box (figure 10.21):

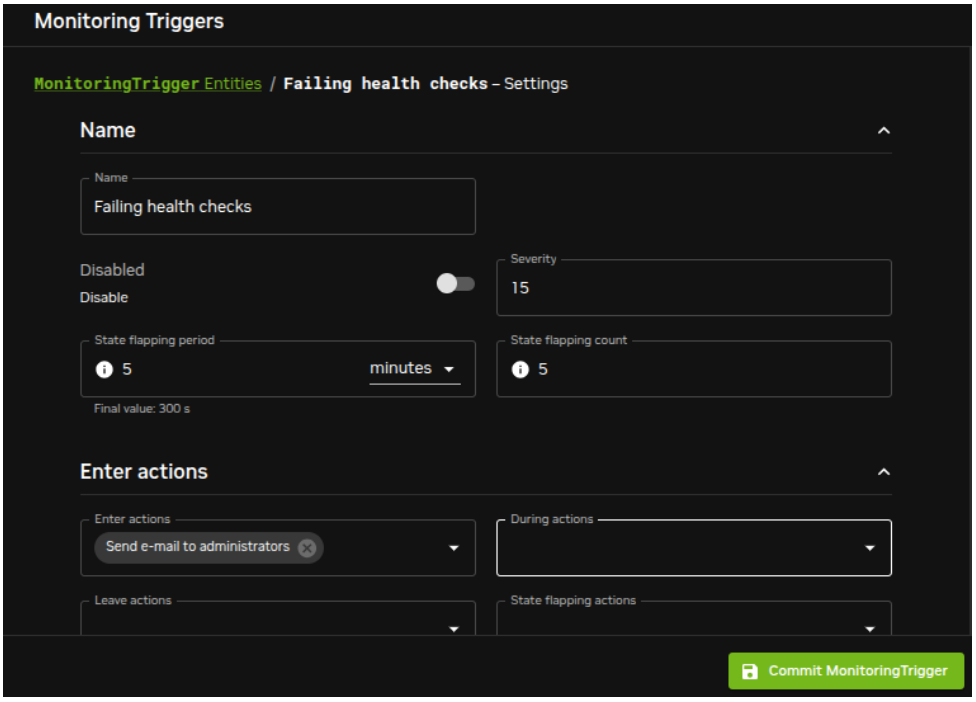


Figure 10.21: Base View Monitoring Configuration: Triggers

and this monitoring trigger configuration should then be committed.

Carrying Out Post-Drain Actions

A special, and hidden, setting for a triggered drain is `post-drain-actions`. This allows one or more actions to be triggered after a node has reached a fully-drained state after a `drain` action.

Example

```
[basecm11->monitoring]% trigger
[basecm11->monitoring->trigger]% add mydrain
[basecm11->...->trigger*[mydrain*]]% set enteractions drain
[basecm11->...->trigger*[mydrain*]]% set -e -v post-drain-actions send\ e-mail\ to\ administrators  poweroff
[basecm11->...->trigger*[mydrain*]]% commit
```

In the preceding example, nodes that enter a drained state, after they are fully-drained, send an e-mail to the administrator and are powered off (page 597 and section G.4.1).

The syntax of the post-drain action is:

```
set -e -v post-drain-actions <action name> [<action name> ...]
```

Adding Custom Triggers: Any Measurable, Any Action

More triggers can be added. The `killallyestrigger` example from the basic example of section 10.1, seen in figures 10.3 and 10.4, is one such example.

The idea is that actions are launched from triggers, and the action for the trigger can be set to a predefined action, or to a custom action.

The Expression Subwindow For A Trigger

When editing a trigger as in figure 10.21 an `Expression` subwindow can be opened. For example, for the `Failing health checks trigger`, the Base View navigation path is:

`Monitoring > Triggers > Failing health checks` > `Expression`

The `Expression` subwindow allows parameters such as entity, measurable, parameters, (comparison) operator, and measurable value be set for a trigger (figure 10.22):

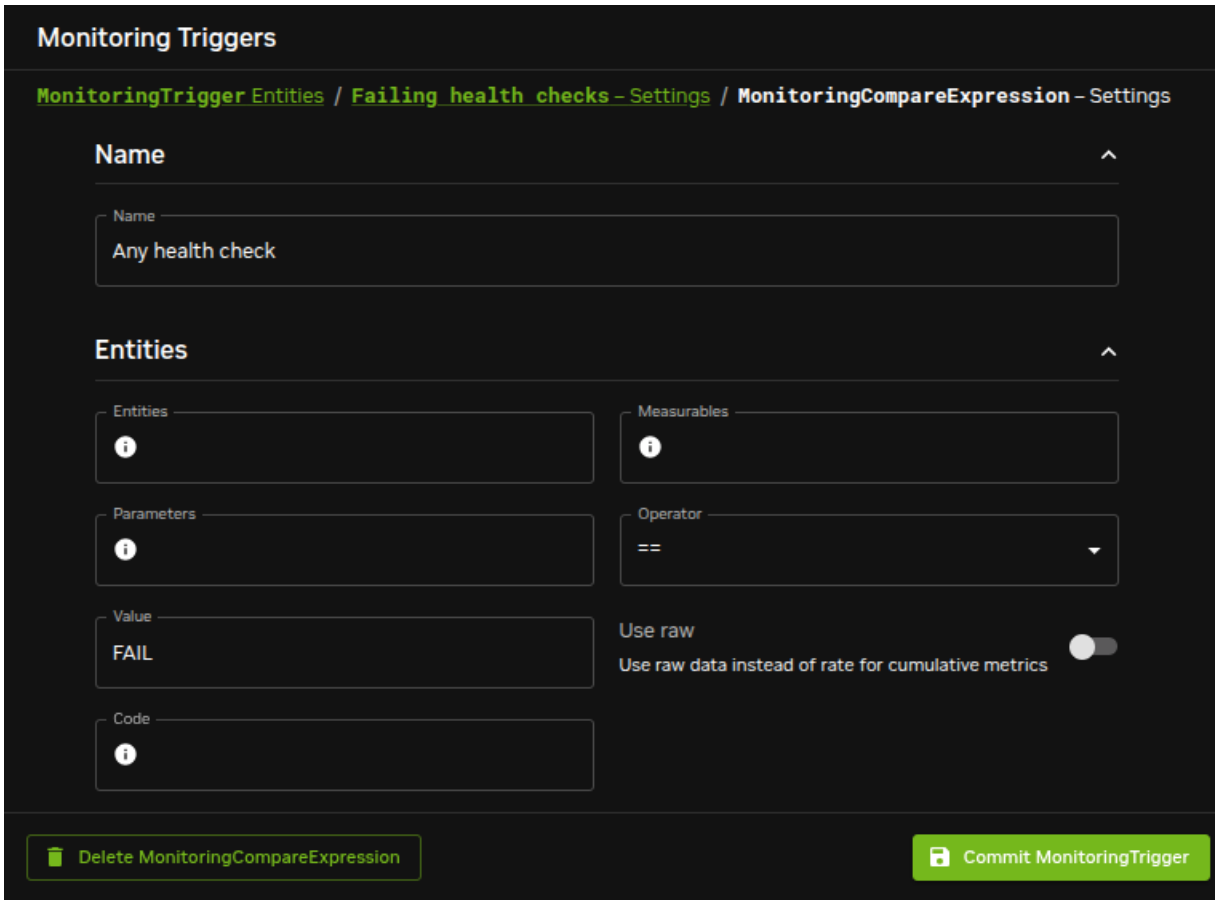


Figure 10.22: Base View Monitoring Configuration: Triggers Expression

The trigger launch is carried out when, during sampling, CMDaemon evaluates the expression as being true.

An example `cmsh` session to set up an expression for a custom trigger might be as follows, where the administrator is setting up the configuration so that an e-mail is sent by the monitoring system when a

node is detected as having gone down:

Example

```
[root@basecm11 ~]# cmsh
[basecm11]% monitoring trigger add nodedown
[basecm11->monitoring->trigger*[nodedown*]]% expression
[basecm11->monitoring->trigger*[nodedown*]->expression[compare]]% show
Parameter                      Value
------------------------------ ------------------------------------------------
Name                           compare
Revision
Type                           MonitoringCompareExpression
Entities
Measurables
Parameters
Operator                       ==
Value                          FAIL
Use raw                        no
Code                           <0B>
[basecm11->monitoring->trigger*[nodedown*]->expression*[compare*]]% set value down
[basecm11->monitoring->trigger*[nodedown*]->expression*[compare*]]% set operator eq
[basecm11->monitoring->trigger*[nodedown*]->expression*[compare*]]% set measurables devicestate
```

To add a touch of realism, a deliberate mistake is set here—the use of `devicestate` (the data producer) instead of `devicestatus` (the measurable). The `validate` command (page 66) gives a helpful warning here, so that the cluster administrator can fix the setting:

```
[basecm11->monitoring->trigger*[nodedown*]->expression*[compare*]]% validate
Field            Message
---------------- ------------------------------------------------------------------------------
actions          Warning: No actions were set
measurables/     Warning: No known measurable matches the specified regexes ('devicestate', '')
parameters
[basecm11->monitoring->trigger*[nodedown*]->expression*[compare*]]% set measurables devicestatus
[basecm11->monitoring->trigger*[nodedown*]->expression*[compare*]]% ..;..
[basecm11->monitoring->trigger*[nodedown*]]% set enteractions send e-mail to administrators
[basecm11->monitoring->trigger*[nodedown*]]% commit
```

10.4.6 Monitoring Configuration: Health status

The `Health status` window (figure 10.23) displays all the nodes, and summarizes the results of all the health checks that have been run against them over time, by presenting a table of the associated severity levels (section 10.2.7):

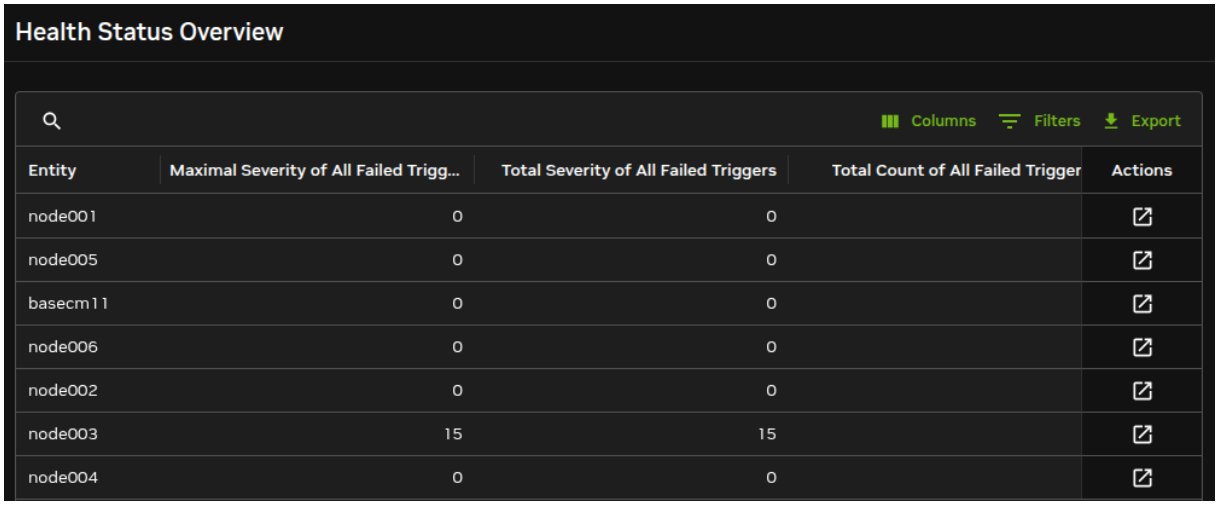


Figure 10.23: Base View Monitoring Configuration: Health Status

In the example shown in figure 10.23 the entity `node003` shows a severity issue, while the other devices are fine. Details of the individual health checks per entity can be viewed by clicking on its associated icon, which opens up a subwindow (figure 10.24). For this example the issue turns out to be due to a `Fail` status in the `ldap` measurable.

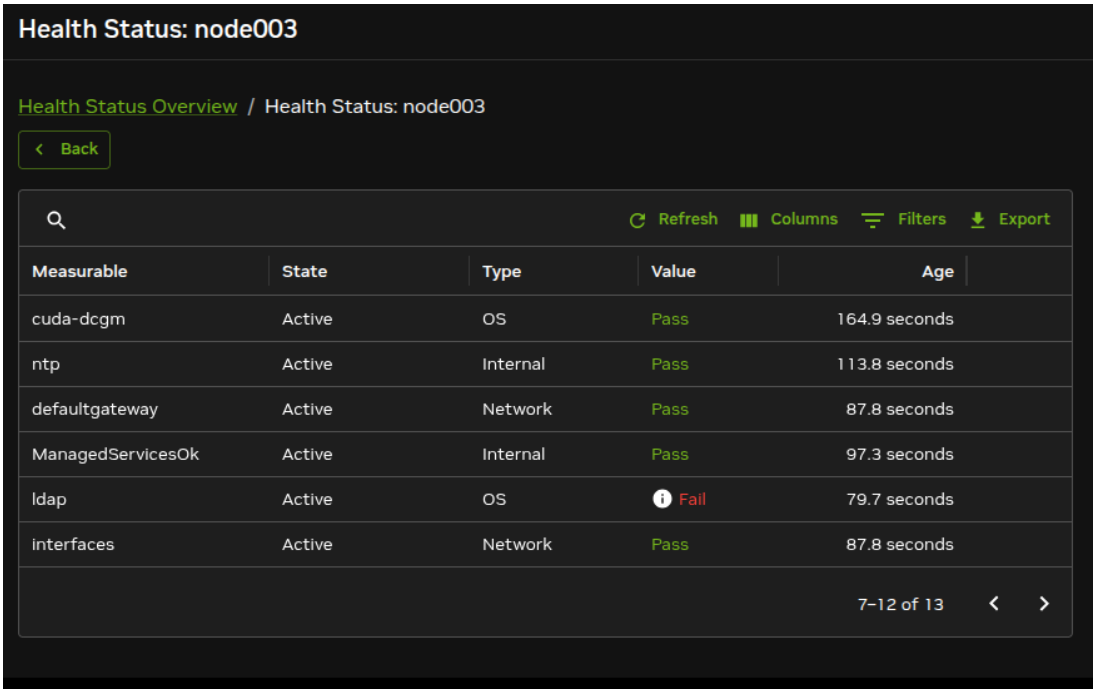


Figure 10.24: Base View Monitoring Configuration: Health Status For An Entity

10.4.7 Monitoring Configuration: All Health Checks

The `All Health checks` window (figure 10.25) shows all the running health checks for all entities.

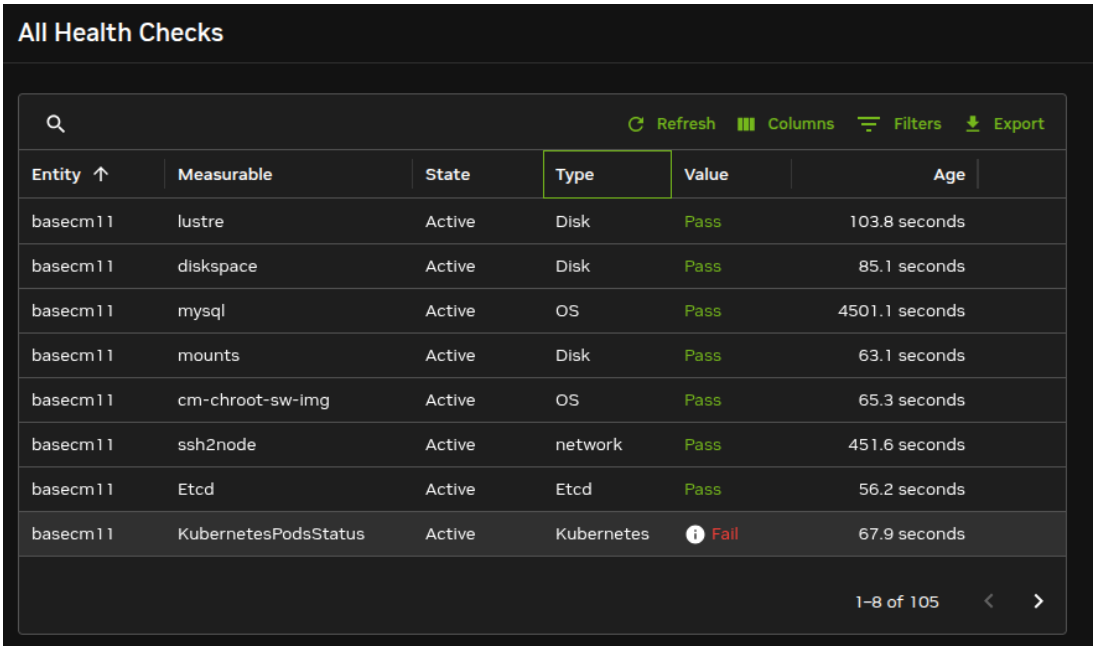


Figure 10.25: Base View Monitoring Configuration: All Health Checks For All Entities

The `Filters` option at the top of the table can be used to show the results per entity, measurable, state, or type only. If the entity filter is set to `node003`, which is as set in section 10.4.6, figure 10.24, then the results table looks similar to the table there.

10.4.8 Monitoring Configuration: Standalone Monitored Entities

The `Standalone Monitored Entities` window allows the cluster administrator to define a standalone entity. A standalone entity is one that is not managed by BCM—which means that no CMDaemon is running on it to gather data and for managing it—but the entity can still be monitored. For example, a workstation that is running the Base View browser could be the standalone entity. This could have its connectivity monitored by pinging it from the head node with a custom script.

10.4.9 Monitoring Configuration: PromQL Queries

The `Prometheus Query list` window displays the list of PromQL job-related queries, and allows query properties to be edited.

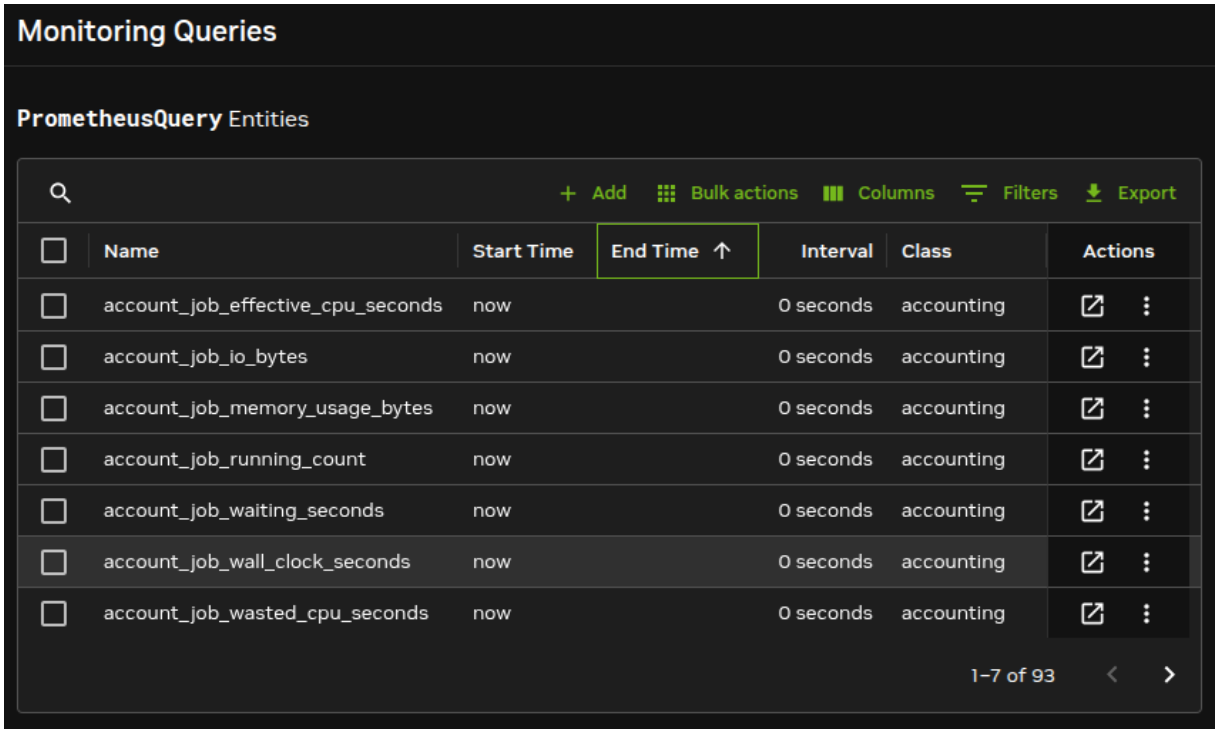


Figure 10.26: Base View Monitoring Configuration: PromQL Queries

Drilldowns can also be viewed using the Base View navigation path:

`Monitoring` > `Queries` > *specific Prometheus query* > `Manage the drilldown queries`

PromQL job queries are discussed further in section 12.3.

10.5 The `monitoring` Mode Of `cmsh`

This section covers how to use `cmsh` to configure monitoring. The `monitoring` mode in `cmsh` corresponds generally to the `Monitoring` resource of Base View in section 10.4. Similarly to how monitoring subwindows are accessed in Base View, the `monitoring` mode of `cmsh` is itself is not used directly, except as a way to access the monitoring configuration submodes of `cmsh`.

For this section some familiarity is assumed with handling of objects as described in the introduction to working with objects (section 2.5.3). When using `cmsh`'s monitoring mode, the properties of objects in the submodes are how monitoring settings are carried out.

The `monitoring` mode of `cmsh` gives access to 9 modes under it:

Example

```
[root@myheadnode ~]# cmsh
[myheadnode]% monitoring help | tail -11
============================== Monitoring ================================
action ........................ Enter action mode
consolidator .................. Enter consolidator mode
labeledentity ................. Enter labeled entity mode
measurable .................... Enter measurable mode
query.......................... Enter monitoring query mode
report......................... Enter report mode
setup ......................... Enter monitoring configuration setup mode
standalone .................... Enter standalone entity mode
```

```
trigger ..................... Enter trigger mode
```

For convenience, a tree of modes for monitoring submodes is shown in figure 10.27.

```
monitoring
      ├── action (section 10.5.1)
      ├── consolidator (section 10.5.2)
      │   └── consolidators (page 607)
      ├── labeledentity (section 12.2)
      ├── measurable (section 10.5.3)
      ├── query (section 12.3)
      │   └── drilldown
      ├── report
      ├── setup (section 10.5.4)
      │   ├── executionmultiplexers (page 621)
      │   ├── jobmetricsettings (page 623)
      │   ├── nodeexecutionfilters (page 617)
      │   └── dpusettings
      ├── standalone (section 10.5.5)
      └── trigger (section 10.5.6)
          └── expression (page 624)
```

Figure 10.27: Submodes Under monitoring Mode

Sections 10.5.1–10.5.6 give examples of how objects are handled under these monitoring modes. To avoid repeating similar descriptions, section 10.5.1 is relatively detailed, and is often referred to by the other sections.

10.5.1 The action Submode

The action submode under the monitoring mode of cmsh allows monitoring actions to be configured. This mode in cmsh corresponds to the Base View navigation path:

```
Monitoring > Actions
```

described earlier in section 10.4.4:

The action mode handles action objects in the way described in the introduction to working with objects (section 2.5.3). A typical reason to handle action objects—the properties associated with an action script or action built-in—might be to view the actions available, or to add a custom action for use by, for example, a metric or health check.

Some examples of how the action mode is used are now given.

The action Submode: list, show, And get

The list command by default lists the names and properties of actions available from action mode in a table:

Example

```
[myheadnode]% monitoring action
[myheadnode->monitoring->action]% list
Type             Name (key)        Run on            Action
---------------- ----------------  ----------------- -------------------------------------------------
Drain            Drain             Active head node  Drain node (node refuses new WLM jobs)
E-Mail           Send e-mail to    Monitoring node   Send e-mail
```

```
                 administrators
Event            Event             Monitoring node   Send an event to users with connected client
ImageUpdate      ImageUpdate       Active head node  Update the image on the node
NVSMHealthDump   NVSM health dump  Active head node  NVSM health dump
PowerOff         PowerOff          Active head node  Power off a device
PowerOn          PowerOn           Active head node  Power on a device
PowerReset       PowerReset        Active head node  Power reset a device
Reboot           Reboot            Node              Reboot a node
Script           killprocess       Node              /cm/local/apps/cmd/scripts/actions/killprocess.pl
Script           remount           Node              /cm/local/apps/cmd/scripts/actions/remount
Script           testaction        Node              /cm/local/apps/cmd/scripts/actions/testaction
Shutdown         Shutdown          Node              Shutdown a node
Undrain          Undrain           Active head node  Undrain node (node accepts new WLM jobs)
```

The preceding shows the actions available on a newly-installed system.

Instead of using `list`, a convenient way to view the possible actions is to use the `show` command with tab-completion suggestions:

Example

```
[myheadnode->monitoring->action]% show<TAB><TAB>
drain         killprocess    powerreset   send\ e-mail\ to\ administrators      undrain
event         poweroff       reboot       shutdown
imageupdate   poweron        remount      testaction
```

The `show` command of `cmsh` when used with a specific action displays the individual parameters and values of that action:

Example

```
[myheadnode->monitoring->action]% show poweroff
Parameter                      Value
------------------------------ --------------------------------
Action                         Power off a device
Revision
Type                           PowerOff
Name                           PowerOff
Run on                         Active head node
Allowed time
Disable                        no
Suppressed by going down       no
```

The `get` command returns the value of an individual parameter of the action object:

Example

```
[myheadnode->monitoring->action]% get poweroff runon
Active head node
```

The `action` Submode: `add`, `use`, `remove`, `commit`, `refresh`, `modified`, `set`, `clear`, And `validate`

In the basic example of section 10.1, in section 10.1.2, the `killallyes` action was cloned from a similar script using a `clone` option in Base View.

The equivalent can be done with a `clone` command in `cmsh`. However, using the `add` command instead, while it requires more steps, makes it clearer what is going on. This section therefore covers adding the `killallyes` script of section 10.1.2 using the `add` command.

When add is used: an object is added, the object is made the current object, and the name of the object is set, all at the same time. After that, set can be used to set values for the parameters within the object, such as a path for the value of the parameter command.

Adding an action requires that the type of action be defined. Just as tab-completion with show comes up with action suggestions, in the same way, using tab-completion with add comes up with type suggestions.

Running the command help add in the action mode also lists the possible types. These types are drain, e-mail, event, imageupdate, nvsmhealthdump, poweroff, poweron, powerreset, reboot, script, servicerestart, servicestart, servicestop, shutdown, undrain.

The syntax for the add command takes the form:

add <*type*> <*action*>

If there is no killallyes action already, then the name is added in the action mode with the add command, and the script type, as follows:

Example

```
[myheadnode->monitoring->action]% add script killallyes
[myheadnode->monitoring->action*[killallyes*]]%
```

Using the add command drops the administrator into the killallyes object level, where its properties can be set. A successful commit means that the action is stored in CMDaemon.

The converse to the add command is the remove command, which removes an action that has had the commit command successfully run on it.

The refresh command can be run from outside the object level, and it removes the action if it has not yet been committed.

The use command is the usual way of "using" an object, where "using" means that the object being used is referred to by default by any command run. So if the killallyes object already exists, then use killallyes drops into the context of an already existing object (i.e. it "uses" the object).

The set command sets the value of each individual parameter displayed by a show command for that action. The individual parameter script can thus be set to the path of the killallyes script:

Example

```
[...oring->action*[killallyes*]]% set script /cm/local/apps/cmd/scripts/actions/killallyes
```

The clear command can be used to clear the value that has been set for script.

The validate command checks if the object has all required values set to sensible values. So, for example, commit only succeeds if the killallyes object passes validation.

Validation does not check if the script itself exists. It only does a sanity check on the values of the parameters of the object, which is another matter. If the killallyes script does not yet exist in the location given by the parameter, it can be created as suggested in the basic example of section 10.1, in section 10.1.2. In the basic example used in this chapter, the script is run only on the head node. If it were to run on regular nodes, then the script should be copied into the disk image.

The modified command lists changes that have not yet been committed.

10.5.2 The consolidator Submode

Consolidators are introduced in section 10.4.3. Consolidators can be managed in cmsh via the consolidator mode, which is the equivalent of the consolidators window (section 10.4.3) in Base View.

The consolidator mode deals with groups of consolidators. One such pre-defined group is default, while the other is none, as discussed earlier in section 10.4.3:

```
[basecm11->monitoring->consolidator]% list
Name (key)               Consolidators
------------------------ ------------------------
default                  hour
none                     <0 in submode>
```

Each consolidators entry can have its parameters accessed and adjusted.

For example, the parameters can be viewed with:

Example

```
[basecm11->monitoring->consolidator]% use default
[basecm11->monitoring->consolidator[default]]% show
Parameter                      Value
------------------------------ ------------------------------------------------
Name                           default
Revision
Consolidators                  hour
[basecm11->monitoring->consolidator[default]]% consolidators
[basecm11->monitoring->consolidator[default]->consolidators]% list
Name (key)               Interval     Period
------------------------ ------------ ------------
hour                     1h           7y 26w
[basecm11->monitoring->consolidator[default]->consolidators]% use hour
[basecm11->monitoring->consolidator[default]->consolidators[hour]]% show
Parameter                      Value
------------------------------ ------------------------------------------------
Name                           hour
Revision
Maximal age                    0s
Maximal samples                65536
Interval                       1h
Offset                         0s
Kind                           average
Period                         7y 26w
[basecm11->monitoring->consolidator[default]->consolidators[hour]]%
```

For the hour consolidator shown in the preceding example, the number of samples saved per hour can be doubled with:

Example

```
[basecm11->monitoring->consolidator[default]->consolidators[hour]]% set maximalsamples 131072
[basecm11->monitoring->consolidator*[default*]->consolidators*[hour*]]% commit
```

The value of Period is the time for which the samples are kept. Samples older than this are dropped. The rules that are applied to decide how samples are kept are described starting on page 587.

Previously consolidated data is discarded with this type of change, if the number of samples is reduced. Changing parameters should therefore be done with some care.

A new consolidators group can be created if needed.

A Base View way, where a decimalminutes group is created, is discussed in the example in section 10.4.3, page 592.

A cmsh way, where a max-per-day group is created, is discussed in the following section:

Creation Of A Consolidator In cmsh

A new consolidator group, max-per-day, can be added to the default consolidator groups of default and none, with:

Example

```
[basecm11]% monitoring consolidator
[basecm11->monitoring->consolidator]% add max-per-day
[...[max-per-day*]]%
```

Within this new group, a new consolidator item, max-per-day can also be defined. The item can be defined so that it only calculates the maximum value per day. The maximum value option is set using the kind setting, and the per day option is set using the interval setting, since that is the interval with which the old data is consolidated:

Example

```
[...[max-per-day*]]% consolidators
[...[max-per-day*]->consolidators]% add max-per-day
[...[max-per-day*]]% set interval 1d
[...[max-per-day*]]% set kind maximum
[...[max-per-day*]]% show
Parameter                      Value
------------------------------ ----------------------------------
Name                           max-per-day
Revision
Maximal age                    0s
Maximal samples                65536
Interval                       1d
Offset                         0s
Kind                           maximum
Period                         180y 2w
[...[max-per-day*]]% commit
```

The max-per-day consolidator data values are kept for about 180 years.

10.5.3 The measurable Submode

The measurable submode under the monitoring mode of cmsh handles measurable objects, that is: metrics, health checks, and enummetrics. This mode corresponds to the Base View navigation path:

```
Monitoring > Measurables
```

covered earlier in section 10.4.2.

Measurable objects represent the configuration of scripts or built-ins. The properties of the objects are handled in cmsh in the way described in the introduction to working with objects (section 2.5.3).

A typical reason to handle measurable objects might be to view the measurables already available, or to remove a measurable that is in use by an entity.

Measurables cannot be added from this mode. To add a measurable, its associated data producer must be added from monitoring setup mode (section 10.5.4).

This section goes through a cmsh session giving some examples of how this mode is used.

The measurable Submode: list, show, And get

In measurable mode, the list command by default lists the names of all measurable objects along with parameters, their class, and data producer.

Example

```
[basecm11->monitoring->measurable]% list
Type         Name (key)                              Parameter   Class           Producer
------------ --------------------------------------- ----------- --------------- ----------------------
Enum         DeviceStatus                                        Internal        DeviceState
Enum         sys_class_net_addr_assign_type          ens3        System/Network  sample_sys_class_net
...
Enum         sys_class_net_name_assign_type          eth0        System/Network  sample_sys_class_net
Enum         sys_class_net_name_assign_type          eth1        System/Network  sample_sys_class_net
HealthCheck  ManagedServicesOk                                   Internal        CMDaemonState
HealthCheck  TotalPowerShelfCriticalPSU                          Total           AggregatePowerShelf
...
HealthCheck  oomkiller                                           OS              oomkiller
HealthCheck  ssh2node                                            network         ssh2node
Metric       AlertLevel                              count       Internal        AlertLevel
Metric       AlertLevel                              maximum     Internal        AlertLevel
Metric       AlertLevel                              sum         Internal        AlertLevel
Metric       AverageCDULiquidDifferentialPressure                Total           AggregateCDU
Metric       AverageCDULiquidFlow                                Total           AggregateCDU
Metric       AverageCDULiquidReturnTemperature                   Total           AggregateCDU
Metric       AverageCDULiquidSupplyTemperature                   Total           AggregateCDU
Metric       AverageCDULiquidSystemPressure                      Total           AggregateCDU
Metric       AveragePowerShelfTemperature                        Total           AggregatePowerShelf
Metric       BlockedProcesses                                    OS              ProcStat
Metric       BufferMemory                                        Memory          ProcMemInfo
Metric       BytesRecv                               ens3        Network         ProcNetDev
Metric       BytesRecv                               eth0        Network         ProcNetDev
...
```

The preceding example shows excerpts from a list of the measurables that can be set for sampling on a newly-installed system. A full list typically contains over 300 items.

The `list` command in `measurable` submode can be run as:

- `list metric`: to display only metrics
- `list healthcheck`: to display only health checks
- `list enum`: to display only enummetrics

The `show` command of the `measurable` submode of `monitoring` mode displays the parameters and values of a specified measurable, such as, for example `CPUUser`, `DeviceStatus`, or `diskspace`:

Example

Example

```
[myheadnode->monitoring->measurable]% show CPUUser
Parameter                      Value
------------------------------ ------------------------------------------------
Revision
Type                           Metric
Name                           CPUUser
Disabled                       no (ProcStat)
Disable triggers               no (ProcStat)
Maximal samples                4,096 (ProcStat)
Maximal age                    0s (ProcStat)
Producer                       ProcStat
Consolidator                   default (ProcStat)
```

```
Parameter
Description                    CPU time spent in user mode
Class                          CPU
Source                         BCM
Introduce NaN                  yes (ProcStat)
Suppressed by going down       no (ProcStat)
Gap                            0 (ProcStat)
Minimum                        0.000000
Maximum                        0.000000
Cumulative                     yes
Integer                        no
Unit                           Jiffies/s
Access                         Public (ProcStat)
Associated user

[myheadnode->monitoring->measurable]% show DeviceStatus
Parameter                      Value
------------------------------ ------------------------------------------------
Revision
Type                           Enum
Name                           DeviceStatus
Disabled                       no (DeviceState)
Disable triggers               no (DeviceState)
Maximal samples                4,096 (DeviceState)
Maximal age                    0s (DeviceState)
Producer                       DeviceState
Consolidator                   default (DeviceState)
Parameter
Description                    The device status
Class                          Internal
Source                         BCM
Introduce NaN                  no (DeviceState)
Suppressed by going down       no (DeviceState)
Gap                            0 (DeviceState)
Access                         Public (DeviceState)
Associated user

Parameter                      Value
------------------------------ ------------------------------------------------
Revision
Type                           HealthCheck
Name                           diskspace
Disabled                       no (diskspace)
Disable triggers               no (diskspace)
Maximal samples                4,096 (diskspace)
Maximal age                    0s (diskspace)
Producer                       diskspace
Consolidator                   default (diskspace)
Parameter
Description                    checks free disk space
Class                          Disk
Source                         BCM
Introduce NaN                  yes (diskspace)
Suppressed by going down       no (diskspace)
Gap                            0 (diskspace)
Access                         Public (diskspace)
```

```
Associated user
```

The Gap setting here is a number. It sets how many samples are allowed to be missed before a value of NaN is set for the value of the metric.

As detailed in section 10.5.1, tab-completion suggestions for the show command suggest the names of objects that can be used, with the use command in this mode. For show in measurable mode, tab-completion suggests over 300 possible objects:

Example

```
[basecm11->monitoring->measurable]% show
Display all 316 possibilities? (y or n)
AlertLevel:count                          MemoryUsed:cmd                UdpInDatagrams
AlertLevel:maximum                        MemoryUtilization             UdpInErrors
AlertLevel:sum                            MergedReads:vda               UdpNoPorts
AverageCDULiquidDifferentialPressure      MergedReads:vdb               UniqueUserLogin
AverageCDULiquidFlow                      MergedWrites:vda              Uptime
AverageCDULiquidReturnTemperature         MergedWrites:vdb              UsedFiles:/
AverageCDULiquidSupplyTemperature         NVLinkSwitchesClosed          UsedFiles:/boot/efi
AverageCDULiquidSystemPressure            NVLinkSwitchesDown            UsedSpace:/
AveragePowerShelfTemperature              NVLinkSwitchesTotal           UsedSpace:/boot/efi
BlockedProcesses                          NVLinkSwitchesUp              UserTime:cmd
BufferMemory                              NodesClosed                   Utilization:cmd
BytesRecv:eth0                            NodesDown                     VirtualMemoryUsed:cmd
BytesRecv:eth1                            NodesTotal                    WriteTime:vda
BytesSent:eth0                            NodesUp                       WriteTime:vdb
BytesSent:eth1                            OOM\ kill                     Writes:vda
CPUGuest                                  OccupationRate                Writes:vdb
...
```

The single colon (":") indicates an extra parameter for that measurable.

Grepping a metrics list: is sometimes handy because there are a large number of metrics. When listing and grepping, it is usually a good idea to allow for case, and be aware of the existence of the parameter column. For example, the AlertLevel metric is displayed as AlertLevel in the tab-completion suggestions of the show command of the previous example, and in the output of the list command. The has command (explained shortly) tab-completion suggests alertlevel in lowercase. There are also several parameters associated with the AlertLevel command. So using the case-insensitive -i option of grep, and using the head command to display the headers is handy:

Example

```
[basecm11->monitoring->measurable]% list | head -2 ; list metric | grep -i alertlevel
type          name (key)           parameter  class               producer
------------- -------------------- ---------- ------------------- ---------------
Metric        AlertLevel           count      Internal            AlertLevel
Metric        AlertLevel           maximum    Internal            AlertLevel
Metric        AlertLevel           sum        Internal            AlertLevel
```

The get command returns the value of an individual parameter of a particular health check object:

Example

```
[myheadnode->monitoring->measurable]% get oomkiller description
Checks whether oomkiller has come into action (then this check returns FAIL)
[myheadnode->monitoring->measurable]%
```

The measurable Submode: The has Command

The has command is used with a measurable to list the entities that use the measurable. Typically these are nodes, but it can also be other entities, such as the base partition, or a rack.

Example

```
[basecm11->monitoring->measurable]% has alertlevel:sum
basecm11
node001
node002
[basecm11->monitoring->measurable]% use devicesup
[basecm11->monitoring->measurable[DevicesUp]]% has
rack1
base
```

The remaining commands in measurable mode, such as use, remove, commit, refresh, modified, set, clear, and validate; all work as outlined in the introduction to working with objects (section 2.5.3). More detailed usage examples of these commands within a monitoring mode are given in the earlier section covering the action submode (section 10.5.1).

The measurable Submode: An Example Session On Viewing And Configuring A Measurable

A typical reason to look at metrics and health check objects—the properties associated with the script or built-in—might be, for example, to view the operating sampling configuration for an entity.

This section goes through a cmsh example session under monitoring mode, where the setup submode (page 614) is used to set up a health check. The healthcheck can then be viewed from the measurable submode.

In the basic example of section 10.1, a trigger was set up from Base View to check if the CPUUser metric was above 50 jiffies/s, and if so, to launch an action.

A functionally equivalent task can be set up by creating and configuring a health check, because metrics and health checks are so similar in concept. This is done here to illustrate how cmsh can be used to do something similar to what was done with Base View in the basic example. A start is made on the task by creating a health check data producer, and configuring its measurable properties. using the setup mode under the monitoring mode of cmsh. The task is completed in the section on the setup mode in section 10.5.4.

To start the task, cmsh's add command is used, and the type is specified, to create the new object:

Example

```
[root@myheadnode ~]# cmsh
[myheadnode]% monitoring setup
[myheadnode->monitoring->setup]% add healthcheck cpucheck
[myheadnode->monitoring->setup*[cpucheck*]]%
```

The show command shows the parameters.

The values for description, runinbash, script, and class should be set:

Example

```
[...->setup*[cpucheck*]]% set script /cm/local/apps/cmd/scripts/healthchecks/cpucheck
[...->setup*[cpucheck*]]% set description "CPUuser under 50%?"
[...->setup*[cpucheck*]]% set runinbash yes
[...->setup*[cpucheck*]]% set class OS
[...->setup*[cpucheck*]]% commit
[myheadnode->monitoring->setup[cpucheck]]%
```

On running commit, the data producer cpucheck is created:

Example

```
[myheadnode->monitoring->setup[cpucheck]]% exit; exit
[myheadnode->monitoring]% setup list | head -2; setup list | grep -i cpucheck
Type           Name (key)   Arguments    Measurables      Node execution filters
-------------- ------------ ------------ ---------------------------------------
HealthCheck    cpucheck     1 / 222      <0 in submode>
```

The measurable cpucheck, is also created:

Example

```
[myheadnode->monitoring]% measurable list | head -2 ; measurable list | grep -i cpuchec
Type           Name (key)   Parameter  Class    Producer
-------------- ------------ ---------- -------- ---------------
HealthCheck    cpucheck                OS       cpucheck
```

Since the cpucheck script does not yet exist in the location given by the parameter script, it needs to be created. One ugly bash script that can do a health check is:

```
#!/bin/bash

## echo PASS if CPUUser < 50
## cpu is a %, ie: between 0 and 100

cpu=`mpstat 1 1 | tail -1 | awk '{print $3}'`
comparisonstring="$cpu"" < 50"

if (( $(bc <<< "$comparisonstring") )); then
  echo PASS
else
  echo FAIL
fi
```

The script should be placed in the location suggested by the object, /cm/local/apps/cmd/scripts/healthchecks/cpucheck, and made executable with a chmod 700.

The cpucheck object is handled further within the cmsh monitoring setup mode in section 10.5.4 to produce a fully-configured health check.

10.5.4 The setup Submode

The setup Submode: Introduction

The setup submode under the monitoring mode of cmsh allows access to all the data producers. This mode in cmsh corresponds to the Base View navigation path:

Monitoring > Data Producers

covered earlier in section 10.4.1.

The setup Submode: Data Producers And Their Associated Measurables

The list of data producers in setup mode should not be confused with the list of measurables in measurable mode. Data producers are not the same as measurables. Data producers produce measurables, although it is true that the measurables are often named the same as, or similar to, their data producer.

In cmsh, data producers are in the Name (key) column when the list command is run from the setup submode:

Example

```
[basecm11->monitoring->setup]% list
Type                    Name (key)           ... Measurables                                Node execution filters
----------------------- ---------------------... ---------------------------------------- ----------------------
AggregateCDU            AggregateCDU         ... 7: AverageCDULiquidDifferentialPressure,+ <1 in submode>
AggregateNode           AggregateNode        ... 25: ManagedServicesOk:fail, ManagedServi+ <1 in submode>
AggregatePDU            AggregatePDU         ... 2: PhaseLoad, TotalInletActive            <1 in submode>
AggregatePowerCircuit   AggregatePowerCircuit... 6: TotalCircuitCurrent, TotalCircuitCurr+ <1 in submode>
AggregatePowerShelf     AggregatePowerShelf  ... 11: AveragePowerShelfTemperature, TotalP+ <1 in submode>
AggregateSwitch         AggregateSwitch      ... 6: TotalBytesRecv, TotalBytesSent, Total+ <1 in submode>
AlertLevel              AlertLevel           ... 3: AlertLevel:count, AlertLevel:maximum,  <1 in submode>
CMDaemonState           CMDaemonState        ... 1: ManagedServicesOk                      <0 in submode>
CPU                     CPUSampler           ... 0:                                        <0 in submode>
...
```

In the preceding example, the `AlertLevel` data producer has 3 showing at the start of the measurables column. This means that this `AlertLevel` data producer provides 3 measurables.

If the data producer is disabled, then the measurables are disabled and no sampling is carried out for those measurables.

Measurables must exist in the data producer object—that is, the value in the the `Measurables` column must be more than 0—and the measurable must also be enabled to allow data sampling and collection to take place. If a measurable exists, then rules—set by node execution filters (page 617) and execution multiplexers (page 621)—determine where the data sampling is carried out and where collection happens.

To clarify this point: if the `list` command is run from `setup` mode to list data producers, then the producers that have configured measurables are the ones that have 1 or more as a value in the `Measurables` column. Conversely, the data producers with 0 in the `Measurables` column have no configured measurables, whether enabled or disabled, which means that the measurables are effectively just placeholders until the software for the measurables data producers is installed.

The list of measurables can be viewed in `setup` mode and `measurable` mode. For example, the three `AlertLevel` measurables produced by the `AlertLevel` producer can be seen in the modes with:

Example

```
[basecm11->monitoring->setup]% list alertlevel
Type         Name (key)  ... Measurables                                                Node execution filters
------------ ----------- ... ---------------------------------------------------------- ----------------------
AlertLevel   AlertLevel  ... 3: AlertLevel:count, AlertLevel:maximum, AlertLevel:sum  <1 in submode>
[basecm11->monitoring->setup]% get alertlevel measurables
AlertLevel:count
AlertLevel:maximum
AlertLevel:sum
[basecm11->monitoring->setup]% measurable
[basecm11->monitoring->measurable]% list | head -2; list | grep -i alertlevel
Type         Name (key)                Parameter        Class           Producer
------------ ------------------------- ---------------- --------------- ----------------
Metric       AlertLevel                count            Internal        AlertLevel
Metric       AlertLevel                maximum          Internal        AlertLevel
Metric       AlertLevel                sum              Internal        AlertLevel
[basecm11->monitoring->measurable]%
```

Similarly, the non-existence of the `CPUSampler` measurable can be viewed in `setup` mode and `measurable` modes with:

Example

```
[basecm11->monitoring->setup]% list cpu
Type         Name (key)  ... Measurables  Node execution filters
------------ ----------- ... ------------ ----------------------
CPU          CPUSampler  ... 0:           <0 in submode>
[basecm11->monitoring->setup]% get cpusampler measurables

[basecm11->monitoring->setup]% measurable
[basecm11->monitoring->measurable]% list | head -2; list | grep -i cpusampler
Type         Name (key)                 Parameter        Class           Producer
------------ -------------------------- ---------------- --------------- ----------------
```

The `setup` Submode: Listing Nodes That Use A Data Producer

The `nodes` command can be used to list the nodes on which a data producer <*data producer*> runs. It is run in the `setup` submode level of the `monitoring` mode as:

`nodes` <*data producer*>

Example

```
[basecm11->monitoring->setup]% list | head -2; list | grep mount
Type          Name (key)  Arguments    Measurables  Node execution filters
------------- ----------- ------------ ------------ ----------------------
HealthCheck   mounts                    1: mounts    <0 in submode>
[basecm11->monitoring->setup]% nodes mounts
node001..node003,basecm11
```

The `setup` Submode: Data Producers Properties

Any data producer from the full list in `setup` mode can, if suitable, be used to provide a measurable for any entity.

An example is the data producer `AlertLevel`. Its properties can be seen using the `show` command:

Example

```
[basecm11->monitoring->setup]% show alertlevel
Parameter                      Value
------------------------------ ------------------------------------------------------
Measurables                    3: AlertLevel:count, AlertLevel:maximum, AlertLevel:sum
Revision
Type                           AlertLevel
Name                           AlertLevel
Maximal age                    0s
Maximal samples                4096
Startup delay                  0s
Consolidator                   default
Disabled                       no
Disable triggers               no
Only when idle                 no
Maximal measurables            512
When                           Timed
Pre job                        no
Post job                       no
Interval                       2m
Offset                         1m
Gap                            0
Fuzzy offset                   0
```

```
Automatic reinitialize          yes
Introduce NaN                   yes
Suppressed by going down        no
Description                     Alert level as function of all trigger severities
Notes                           <0B>
Access                          Public
Associated user
Node execution filters          <1 in submode>
Execution multiplexer           <1 in submode>
[basecm11->monitoring->setup]%
```

These properties are described in section 10.4.1. Most of these properties are inherited by the measurables associated with the data producer, which in the AlertLevel data producer case are alertlevel:count, alertlevel:maximum, and alertlevel:sum.

The setup Submode: Deeper Submodes

One level under the setup submode of monitoring mode are 4 further submodes (modes deeper than submodes are normally also just called submodes for convenience, rather than sub-submodes):

- nodeexecutionfilters
- executionmultiplexers
- jobmetricsettings
- dpusettings

Node execution filters: A way to filter execution (restrict execution) of the data producer.

If no node execution filter is set for that data producer, then the data producer runs on all nodes of the cluster. Filters can be of the type:

- category
- lua
- node
- overlay
- resource
- type

The type is set when the filter is created.

- **The nodes command for listing the execution nodes**

  Running the nodes command for a data producer lists which nodes the execution of the data producer is run on.

  Example

```
[myhost->monitoring->setup]% nodes procmeminfo
mon001..mon003,myhost,osd001,osd002,node001,node002
[myhost->monitoring->setup]% nodes ssh2node
myhost
[myhost->monitoring->setup]% nodes devicestate
myhost
[myhost->monitoring->setup]% foreach * ( get name; nodes) | paste - - | sort
```

```
AggregateCDU    myhost
AggregateNode   myhost
AggregatePDU    myhost
       ...[so far the data producers run only on the head node, but eventually see other nodes]...
CMDaemonState   node001..node006,myhost
CPUSampler      node001..node006,myhost
DeviceState     myhost
        ...[in the next entry, the nodes command shows that the measurable is not used:]...
Docker          Not used
...
```

Most of the default data producers that are used by the cluster run on an active head node, and often on the regular nodes.

- `nodexecutionfilters` **to restrict data producer execution**

  The `rogueprocess` (page 1039) data producer is one of the few that by default runs on a regular node. Restricting a data producer to run on a particular list of nodes can be carried out as follows on a cluster that is originally in its default state:

  Example

```
[basecm11->monitoring->setup[rogueprocess]]% nodeexecutionfilters
[basecm11->monitoring->setup[rogueprocess]->nodeexecutionfilters]% add<TAB><TAB>
category  lua       node      overlay   resource  type
[...tup[rogueprocess]->nodeexecutionfilters]% add node justthese
[...tup*[rogueprocess*]->nodeexecutionfilters*[justthese*]]% show
Parameter                      Value
------------------------------ ------------------------------------------------
Name                           justhese
Revision
Type                           Node
Filter operation               Include
Nodes
[...tup*[rogueprocess*]->nodeexecutionfilters*[justthese*]]% set nodes node001,node002
[...tup*[rogueprocess*]->nodeexecutionfilters*[justthese*]]% show
Parameter                      Value
------------------------------ ------------------------------------------------
Name                           justhese
Revision
Type                           Node
Filter operation               Include
Nodes                          node001,node002
[...tup*[rogueprocess*]->nodeexecutionfilters*[justthese*]]% commit
```

  This way, the `rogueprocess` health check runs on just those nodes (`node001`, `node002`), and none of the others.

- **Restricting a data producer execution to the head node—monitoring a process on the head node**

  Another example of data producer restriction is as follows: an administrator may wish to monitor the `slapd` process on the head nodes. In `cmsh`, this can be achieved by creating a new data producer, `slapd` of type `procpidstat`.

  and other properties as parameters to the measurables of the `procpidstat` data producer, as in the following session:

Example

```
[basecm11->monitoring->setup]% add procpidstat slapd
[basecm11->monitoring->setup*[slapd*]]% set process slapd
[basecm11->monitoring->setup*[slapd*]]% set consolidator none; commit
[basecm11->monitoring->setup[slapd]]%
```

Defining slapd as the process name for the data producer means that the following procpidstat measurables become defined:

Example

```
[basecm11->monitoring->setup[slapd]]% get measurables
MemoryUsed:slapd
RunTime:slapd
SystemTime:slapd
ThreadsUsed:slapd
UserTime:slapd
Utilization:slapd
VirtualMemoryUsed:slapd
[basecm11->monitoring->setup[slapd]]% get measurables
```

The nodeexecutionfilters settings can be set for the slapd data producer as follows:

- One way to set it might be:

  Example

  ```
  [basecm11->monitoring->setup[slapd]]% nodeexecutionfilters
  [basecm11->monitoring->setup[slapd]->nodeexecutionfilters]% add type myheadnode
  [basecm11->monitoring->setup*[slapd*]->nodeexecutionfilters*[myheadnode*]]% show
  Parameter                      Value
  ------------------------------ ------------------------------------------------
  Name                           myheadnode
  Revision
  Type                           Type
  Filter operation               Include
  Head node                      no
  Physical node                  no
  Cloud node                     no
  Lite node                      no
  DPU node                       no
  [basecm11->monitoring->setup*[slapd*]->nodeexecutionfilters*[myheadnode*]]% set headnode yes
  [basecm11->monitoring->setup*[slapd*]->nodeexecutionfilters*[myheadnode*]]% commit; exit
  ```

- Alternatively, in nodeexecutionfilters mode, the type that can be added is one of the names reserved for a node type (Head node, Cloud node Lite node, DPU node) which automatically sets the type parameter to the value of type, and sets node type to the reserved name.

  :

  Example

  ```
  [basecm11->monitoring->setup[slapd]]% nodeexecutionfilters
  [basecm11->monitoring->setup[slapd]->nodeexecutionfilters]% add type headnode
  [basecm11->monitoring->setup*[slapd*]->nodeexecutionfilters*[headnode*]]% get headnode
  yes
  [basecm11->monitoring->setup*[slapd*]->nodeexecutionfilters*[headnode*]]% commit; exit
  ```

The filter now works on all head nodes. To have it work on only the active head node, the `active` command can be used instead, at `nodeexecutionfilter` mode level:

Example

```
[basecm11->monitoring->setup[slapd]->nodeexecutionfilters]% active
Added active resource filter
```

The newly-defined `slapd` parameter metrics can now have its output displayed or plotted just like any other metric:

Example

```
[basecm11->device[basecm11]]% latestmetricdata | head -2; latestmetricdata | grep slapd
Measurable           Parameter    Type            Value       Age        State      Info
-------------------- ------------ --------------- ----------- ---------- ---------- --------
MemoryUsed           slapd        Process         47 MiB      54.8s
RunTime              slapd        Process         10h 40m     54.8s
SystemTime           slapd        Process         1.49s       54.8s
ThreadsUsed          slapd        Process         7           54.8s
UserTime             slapd        Process         4.31s       54.8s
Utilization          slapd        Process         13.04 s/d   54.8s
VirtualMemoryUsed    slapd        Process         900 MiB     54.8s
```

- **Filtering a data producer by resource**

  A data producer can also be set up so that it is run on a particular list of nodes filtered by resource. The resources that are available to a node can be viewed using the command `monitoringresources` for that device:

  Example

```
[basecm11->device[basecm11]]% monitoringresources
Active
ActiveWLMServer
Ethernet
OpenStack Foundation
OpenStack Nova
UBUNTU24_04
backup
boot
firewall
headnode
monitoring
overlay:slurm-accounting
overlay:slurm-server
overlay:wlm-headnode-submit
provisioning
slurmaccounting
slurmserver
slurmsubmit
storage
[basecm11->device[basecm11]]%
```

An example of where running a node execution filter by resource is useful, is for data producers that are intended to run on the active head node. Most data producers that are used by the cluster run on an active head node (besides often running on the regular nodes too).

Thus, for example, the cpucheck health check from page 613 can be set to run on the active head node, by creating an arbitrary resource called myactive:

Example

```
[basecm11->monitoring->setup[cpucheck]]% nodeexecutionfilters
[...tup[cpucheck]->nodeexecutionfilters]% add resource "myactive"
[...tup*[cpucheck*]->nodeexecutionfilters*[myactive*]]% show
Parameter                      Value
------------------------------ ------------------------------------------------
Name                           myactive
Revision
Type                           Resource
Filter operation               Include
Resources
Operator                       OR
```

and then setting the Resources parameter to Active:

```
[...tup*[cpucheck*]->nodeexecutionfilters*[myactive*]]% set resources Active
[...tup*[cpucheck*]->nodeexecutionfilters*[myactive*]]% show
Parameter                      Value
------------------------------ ------------------------------------------------
Name                           myactive
Revision
Type                           Resource
Filter operation               Include
Resources                      Active
Operator                       OR
[...tup*[cpucheck*]->nodeexecutionfilters*[myactive*]]% commit
```

The cpucheck health check then runs on the active head node, whichever head node it is.

When node execution filtering is carried out, the filtered data is not dropped by default. Filtered data can be dropped for a measurable or an entity with the monitoringdrop command (section 10.6.8).

Execution multiplexer: A way to multiplex execution (have execution work elsewhere) for a data producer. The object tells BCM about the entities that the data producer is executing sampling at. Thus:

- For a node execution filter, as explained earlier on page 617, a data producer runs and gathers data at the entity which is defined by the node execution filter. The entity can be of type:
  - category
  - lua
  - node
  - overlay
  - resource
  - type
- For an execution multiplexer, a data producer gathers samples from other entities. The entities are defined into groups called execution multiplexers, and can be of type:

- category
- dynamic (an extra type not in nodeexecutionfilters)
- lua
- node
- overlay
- resource
- type

Execution multiplexers can thus be category multiplexers, dynamic multiplexers, and so on.

The executionmultiplexers mode can be entered for a data producer dmesg with:

Example

```
[root@basecm11 ~]# cmsh
[basecm11]% monitoring setup executionmultiplexers dmesg
```

Running the commands: help add, or help set, can be used to show the valid syntax in this submode.

Most data producers run on a head node, but sample from the regular nodes. So, for example, the dmesg health check from Appendix G.2.1 can be set to sample from the regular nodes by setting it to carry out execution multiplexing to specified node entities using a node multiplexer with the arbitrary name of nodes as follows:

Example

```
[basecm11->monitoring->setup[dmesg]->executionmultiplexers]% add<TAB><TAB>
category  dynamic      lua       node      overlay   resource  type
[basecm11->monitoring->setup[dmesg]->executionmultiplexers]% add node nodes
[basecm11->...*[dmesg*]->executionmultiplexers*[nodes*]]% show
Parameter                      Value
------------------------------ ------------------------------------------------
Name                           nodes
Revision
Type                           Node
Filter operation               Include
Nodes

[basecm11->...*[dmesg*]->executionmultiplexers*[nodes*]]% set nodes node001,node002
[basecm11->...*[dmesg*]->executionmultiplexers*[nodes*]]% show
Parameter                      Value
------------------------------ ------------------------------------------------
Name                           nodes
Revision
Type                           Node
Filter operation               Include
Nodes                          node001,node002
```

If the type is not set to a particular type, then the type parameter is set to type and a types field becomes available. In that case, if the name is then also set to a value that is a type from the reserved list of monitoring types, then the types parameter is automatically populated by that value.

Example

```
[basecm11->monitoring->setup[dmesg]->executionmultiplexers]% add type sometype
[basecm11->...*[dmesg*]->executionmultiplexers*[sometype*]]% show
Parameter                      Value
------------------------------ ------------------------------------------------
Name                           sometype
Revision
Type                           Type
Filter operation               Include
Types

[basecm11->...*[dmesg*]->executionmultiplexers*[sometype*]]% exit
[basecm11->...*[dmesg*]->executionmultiplexers*]% refresh
[basecm11->...[dmesg]->executionmultiplexers]% add type node
[basecm11->...*[dmesg*]->executionmultiplexers*[node*]]% show
Parameter                      Value
------------------------------ ------------------------------------------------
Name                           node
Revision
Type                           Type
Filter operation               Include
Types                          node
```

The reserved list of monitoring types that can be used in the `name` field to automatically populate the `types` can be found by running the `device monitoringtypes -a` command:

Example

```
[basecm11->...*[dmesg*]->executionmultiplexers*[node*]]% device monitoringtypes -a
Category
CloudNode
ComputeNode
Device
Entity
HeadNode
LiteNode
Network
Node
Partition
PhysicalNode
Rack
SoftwareImage
Switch
```

or by viewing the error message when setting a value for `types` incorrectly:

Example

```
[basecm11->...*[dmesg*]->executionmultiplexers*[node*]]% set types incorrectvalue
'incorrectvalue' is invalid: options are: 'Category' 'CloudNode', 'ComputeNode' 'Device' 'Entity' 'HeadNode'
        'LiteNode' 'Network' 'Node' 'Partition' 'PhysicalNode' 'Rack' 'SoftwareImage' 'Switch'
```

The concepts and expected behavior of node execution filters and execution multiplexers are covered in more explicit detail in Appendix L.

Job metrics settings: Job metrics settings are a submode for setting job metric collection options for the `JobSampler` data producer (section 11.4).

10.5.5 The standalone Submode

The standalone submode under the monitoring mode of cmsh allows entities that are not managed by BCM to be configured for monitoring. This mode in cmsh corresponds to the Base View navigation path:

```
Monitoring > Standalone Monitored Entities
```

covered earlier in section 10.4.8.

The monitoring for such entities has to avoid relying on a CMDaemon that is running on the entity. An example might be a chassis that is monitored via a ping script running on the BCM head node.

10.5.6 The trigger Submode

The trigger submode under the monitoring mode of cmsh allows actions to be configured according to the result of a measurable.

This mode in cmsh corresponds to the Base View navigation path:

```
Monitoring > Triggers
```

covered earlier in section 10.4.5.

By default, there are 3 triggers:

Example

```
[basecm11->monitoring->trigger]% list
Name (key)               Expression               Enter actions During actions Leave actions
------------------------ ------------------------ ------------- -------------- -------------
Failing health checks    (*, *, *) == FAIL        Event
Passing health checks    (*, *, *) == PASS        Event
Unknown health checks    (*, *, *) == UNKNOWN     Event
```

Thus, for a passing, failing, or unknown health check, an event action takes place if entering a state change. The default severity level of a passing health check does not affect the AlertLevel value. However, if the failing or unknown health checks are triggered on entering a state change, then these will affect the AlertLevel value.

The trigger Submode: Setting An Expression

In the basic example of section 10.1, a trigger to run the killallyes script was configured using Base View.

The expression that was set for the killallyes script in the basic example using Base View can also be set in cmsh. For example:

Example

```
[basecm11->monitoring->trigger]% add killallyestrigger
[basecm11->monitoring->trigger*[killallyestrigger*]]% show
Parameter                      Value
------------------------------ ------------------------------------------------
Revision
Name                           killallyestrigger
Disabled                       no
Severity                       10
State flapping period          5m
State flapping count           5
Enter actions
During actions
Leave actions
```

```
State flapping actions
Mark entity as failed            yes
Mark entity as unknown           no
Expression                       (*, *, *) == FAIL
[basecm11->monitoring->trigger*[killallyestrigger*]]% expression
[basecm11->monitoring->trigger*[killallyestrigger*]->expression[compare]]% show
Parameter                        Value
-------------------------------- ------------------------------------------------
Revision
Type                             MonitoringCompareExpression
Name                             compare
Entities
Measurables
Parameters
Operator                         ==
Value                            FAIL
Use raw                          no
Code                             <0B>
[basecm11->monitoring->trigger*[killallyestrigger*]->expression[]]% set entities basecm11
[basecm11->monitoring->trigger*[killallyestrigger*]->expression*[*]]% set measurables CPUUser
[basecm11->monitoring->trigger*[killallyestrigger*]->expression*[*]]% set operator GT
[basecm11->monitoring->trigger*[killallyestrigger*]->expression*[*]]% set value 50
[basecm11->monitoring->trigger*[killallyestrigger*]->expression*[*]]% commit
[basecm11->monitoring->trigger*[killallyestrigger*]->expression*[*]]% set name killallyesexp
Field                    Message
------------------------ ------------------------------------------------------
actions                  Warning: No actions were set
============================== killallyestrigger ==============================
[basecm11->monitoring->trigger[killallyestrigger]->expression[killallyesexp]]% exit
[basecm11->monitoring->trigger[killallyestrigger]->expression]% exit
[basecm11->monitoring->trigger[killallyestrigger]]% set enteractions killallyesaction
[basecm11->monitoring->trigger*[killallyestrigger*]]% commit
[basecm11->monitoring->trigger[killallyestrigger]]%
```

The expression format is shown in cmsh as:

(<*entity*>, <*measurable*>, <*parameter*>) <*comparison operator*> <*value*>

Here:

- an entity, as described in section 10.2.1, can be, for example, a node, category, device, or software image. To include more than one entity for the comparison, the alternation (pipe, |) symbol can be used, with double quotes to enclose the expression.

  Example

  ```
  ...[killallyestrigger*]->expression[]]% set entities "basecm11|node001|compute|gpuimage"
  ```

  In the preceding example, the entity compute could be a category, and the entity gpuimage could be a software image.

- a measurable (section 10.2.1) can be a health check, a metric, or an enummetric. For example: CPUUsage. Alternation works for <*measurable*> in a similar way to that for <*entity*>.

- a parameter is a further option to a measurable. For example, the FreeSpace metric can take a mount point as a parameter. Alternation works for <*parameter*> in a similar way to that for <*entity*>.

- the comparison operator can be:

  EQ: equivalent to, displayed as ==

  NE: not equivalent to, displayed as !=

  GT: greater than, displayed as >

  LT: less than, displayed as <

  If the user uses an arithmetic symbol such as > in cmsh as an unescaped entry, then the entry may unintentionally be interpreted by the shell. That is why the two-letter entries are recommended instead for entry, even though when displayed they display as the arithmetic symbols for easier recognition.

- the value can be a string, or a number.

The regex evaluates to TRUE or FALSE. The trigger runs its associated action in the case of TRUE.

The wildcard * implies any entity, measurable, or parameter when used with the appropriate position according to the syntax of the expression format.

Using .* is also possible to match zero or more of any characters.

Some further expression matching examples:

Example

True for any failing health check:

```
(*, *, *) == FAIL
```

Example

True for any nearly full local disk (less than 10MB left):

```
(*, FreeSpace, sd[a-z]) < 10MB
```

Example

True for any cloud node that is too expensive (price more than more than 10$):

```
(.*cnode.*, Price, *) > 10$
```

Example

Excluding node agw001:

```
(^(?!.*agw001).*$, *, *) == FAIL
```

Example

True for any node in the data, gpu, or hpc categories, that has a nearly full local disk (less than 10MB left):

```
(!resource=category:data|category:gpu|category:hpc, FreeSpace, sd[a-z]) < 10MB
```

The unusual syntax in the preceding example is liable to change in future versions.

At the end of section 10.5.3 a script called cpucheck was built. This script was part of a task to use health checks instead of metrics to set up the functional equivalent of the behavior of the basic example of section 10.1. In this section the task is continued and completed as follows:

```
[basecm11->monitoring->trigger]% expression killallyestrigger
[...trigger[killallyestrigger]->expression[killallyesexp]]% get measurables
CPUUser
[...trigger[killallyestrigger]->expression[killallyesexp]]% set measurables cpucheck
[...trigger*[killallyestrigger*]->expression*[killallyesexp*]]% commit
```

10.6 Obtaining Monitoring Data Values

The monitoring data values that are logged by devices can be used to generate graphs using the methods in section 10.3. However, sometimes an administrator would like to have the data values that generate the graphs instead, perhaps to import them into a spreadsheet for further direct manipulation, or to pipe them into a utility such as `gnuplot`.

10.6.1 Getting The List Of Measurables For An Entity: The `measurables`, `metrics`, `healthchecks` And `enummetrics` Commands

The measurables for a specified entity can be seen with the `measurables` command, and the measurable subtypes can be seen with the corresponding measurable subset commands: `metrics`, `healthchecks` and `enummetrics`. The results look quite similar to the results of the `measurable` submode of the `monitoring` mode (section 10.5.3). However, for entities, the measurables are a sublist of the full number of measurables listed in the `measurable` submode, which in turn are only the list of measurables for the data producers that have been enabled.

For example, within `device` mode where the entities are typically the head node and regular nodes, running `metrics` with a specified entity shows only the metrics that are configured for that entity. Thus if the entity is a head node, then only head node metrics are shown; and if the entity is a regular node, only regular node metrics are shown:

Example

```
[basecm11->device]% enummetrics node001
Type  Name                            Parameter  Class           Producer
----- ------------------------------  ---------- --------------- --------------------
Enum  DeviceStatus                               Internal        DeviceState
Enum  sys_class_net_addr_assign_type  ens3       System/Network  sample_sys_class_net
Enum  sys_class_net_name_assign_type  ens3       System/Network  sample_sys_class_net

[basecm11->device]% use basecm11
[basecm11->device[basecm11]]% measurables
Type         Name                           Parameter   Class           Producer
------------ ------------------------------ ---------- --------------- --------------------
Enum         DeviceStatus                   Internal    DeviceState
Enum         sys_class_net_addr_assign_type eth0        System/Network  sample_sys_class_net
Enum         sys_class_net_addr_assign_type eth1        System/Network  sample_sys_class_net
Enum         sys_class_net_name_assign_type eth0        System/Network  sample_sys_class_net
Enum         sys_class_net_name_assign_type eth1        System/Network  sample_sys_class_net
HealthCheck  ManagedServicesOk                          Internal        CMDaemonState
HealthCheck  chrootprocess                              OS              chrootprocess
...
[basecm11->device[basecm11]]% exit
[basecm11->device]% metrics node001
Type         Name               Parameter  Class                        Producer
------------ ------------------ ---------- ---------------------------- ----------------
Metric       AlertLevel         count      Internal                     AlertLevel
Metric       AlertLevel         maximum    Internal                     AlertLevel
Metric       AlertLevel         sum        Internal                     AlertLevel
Metric       BlockedProcesses              OS                           ProcStat
...
```

Typically the number of metrics listed on the head node will differ from those listed on a regular node. Whatever each number is, it cannot be more than the number of metrics seen in the number of metrics listed in the `measurable` submode of section 10.5.3.

The preceding example shows the measurables listing commands being carried out on head nodes and regular nodes. These commands can be used on other entities too. For example, the `base` partition in `partition` mode, where the measurables can be listed with:

Example

```
[basecm11->device]% partition use base
[basecm11->partition[base]]% measurables
Type         Name (key)                            Parameter    Class    Producer
------------ ------------------------------------ ------------ -------- ------------------------
HealthCheck  TotalPowerShelfCriticalPSU                         Total    AggregatePowerShelf
HealthCheck  TotalPowerShelfDegradedPSU                         Total    AggregatePowerShelf
HealthCheck  TotalPowerShelfHealthyPSU                          Total    AggregatePowerShelf
Metric       AverageCDULiquidDifferentialPressure               Total    AggregateCDU
Metric       AverageCDULiquidFlow                               Total    AggregateCDU
Metric       AverageCDULiquidReturnTemperature                  Total    AggregateCDU
Metric       AverageCDULiquidSupplyTemperature                  Total    AggregateCDU
Metric       AverageCDULiquidSystemPressure                     Total    AggregateCDU
Metric       AveragePowerShelfTemperature                       Total    AggregatePowerShelf
Metric       CoresDown                                          Total    ClusterTotal
Metric       CoresTotal                                         Total    ClusterTotal
Metric       CoresUp                                            Total    ClusterTotal
Metric       DPUNodesClosed                                     Total    ClusterTotal
Metric       DPUNodesDown                                       Total    ClusterTotal
Metric       DPUNodesTotal                                      Total    ClusterTotal
Metric       DPUNodesUp                                         Total    ClusterTotal
Metric       DevicesClosed                                      Total    ClusterTotal
Metric       DevicesDown                                        Total    ClusterTotal
...
```

The values for metric samples and health checks can be obtained from within device mode in various ways, and are explained next.

10.6.2 On-demand Metric Sampling And Health Checks

The `samplenow` Command For On-Demand Measurable Samples

An administrator can do live sampling, or sampling on-demand, for specified entities by using the `samplenow` command. The command has the following syntax:

`samplenow [`*OPTIONS*`] [<`*entity*`>] [<`*measurable*`> ...]`

The command can be run without options when an entity object, such as a node is used (output truncated):

Example

```
[basecm11->device]% use basecm11
[basecm11->device[basecm11]]% samplenow
Measurable         Parameter  Type      Value          Age     State  Info
------------------ ---------- --------- -------------- ------- ------ ------
AlertLevel         count      Internal  0              1.19s
AlertLevel         maximum    Internal  0              1.19s
AlertLevel         sum        Internal  0              1.19s
BlockedProcesses              OS        0 processes    1.19s
BufferMemory                  Memory    3.7 MiB        1.2s
BytesRecv          eth0       Network   152.221 B/s    1.2s
...
```

The entity used can also be in other modes that have measurables, such as the base partition (output truncated):

Example

```
[basecm11->device]% partition use base
[basecm11->partition[base]]% samplenow
Measurable         Parameter  Type      Value           Age     State  Info
------------------ ---------- --------- --------------- ------- ------ ------
CoresDown                     Total     0               0.298s
CoresTotal                    Total     4               0.298s
CoresUp                       Total     4               0.298s
DPUNodesClosed                Total     0               0.298s
DPUNodesDown                  Total     0               0.298s
DPUNodesTotal                 Total     0               0.298s
DPUNodesUp                    Total     0               0.298s
DevicesClosed                 Total     0               0.298s
DevicesDown                   Total     0               0.298s
DevicesTotal                  Total     0               0.298s
...
```

The -n|--nodes Option

The -n option in device mode allows specified nodes or node ranges:

Example

```
[basecm11->partition[base]]% device
[basecm11->device]% samplenow -n node001..node002 loadone
Entity       Measurable         Parameter Type      Value   Age     Info
------------ ------------------ --------- --------- ------- ------- -----
node001      LoadOne                      OS        0.04    0.08s
node002      LoadOne                      OS        0       0.077s
```

The --metrics And --checks Option

For a particular entity:

- All metrics can be sampled on demand with the --metrics option
- All health checks can be sampled on demand with the --checks option

Example

```
[basecm11->device]% samplenow --metrics loadone loadfifteen --n node001,node002
Entity       Measurable         Parameter Type      Value  Age     State  Info
------------ ------------------ --------- --------- ------ ------- ------ -----
node001      LoadFifteen                  OS        0      0.01s
node001      LoadOne                      OS        0.14   0.01s
node002      LoadFifteen                  OS        0      0.011s
node002      LoadOne                      OS        0      0.011s

[basecm11->device]% samplenow --checks -n node001..node002
Entity       Measurable         Parameter Type      Value  Age     State  Info
------------ ------------------ --------- --------- ------ ------- ------ -----
node001      ManagedServicesOk            Internal  PASS   0.54s
node001      cuda-dcgm                    OS        PASS   0.54s
node001      defaultgateway               Network   PASS   0.54s
node001      diskspace                    Disk      PASS   0.54s
```

```
node001      dmesg                        OS        PASS   0.54s
node001      interfaces                   Network   PASS   0.54s

[basecm11->device]% samplenow --checks diskspace -n node001..node002
Entity       Measurable         Parameter Type      Value  Age     State  Info
------------ ------------------ --------- --------- ------ ------- ------ -----
node001      diskspace                    Disk      PASS   0.54s
node002      diskspace                    Disk      PASS   0.025s
```

The `--debug` Option

The `--debug` option passes `CMD_DEBUG=1` to the script environment. This can be used to provide extra information on what is happening during sampling.

Example

```
[basecm11->device[node001]]% samplenow ntp
Measurable   Parameter Type      Value  Age     State  Info
------------ --------- --------- ------ ------- ------ -----
ntp                    Internal  PASS   0.136s

[basecm11->device[node001]]% samplenow --debug ntp
Measurable   Parameter Type      Value  Age     State  Info
------------ --------- --------- ------ ------- ------ ----------------------------------------
ntp                    Internal  PASS   0.107s         command: "ps -e"+

[basecm11->device[node001]]% samplenow --debug -v ntp
Measurable   Parameter Type      Value  Age     State  Info
------------ --------- --------- ------ ------- ------ ----------------------------------------
ntp                    Internal  PASS   0.11s          command: "ps -e"
                                                       ntpd process found, pid: 2648
                                                       command: "/bin/ntpq -pn"
                                                       found time syspeer: 10.141.255.254
                                                       send time request to 10.141.255.254
                                                       received a reply from 10.141.255.254
                                                       time from 10.141.255.254 :  1768502974.194
                                                       time on node             :  1768502974.194
                                                       time difference          :           0.000
                                                       execution time 0.02

[basecm11->device[node001]]% !ssh node001 "systemctl stop ntp"
[basecm11->device[node001]]% samplenow --debug -v ntp
Measurable   Parameter Type      Value  Age     State  Info
------------ --------- --------- ------ ------- ------ ----------------------------------------
ntp                    Internal  FAIL   0.107s         command: "ps -e"
                                                       ntpd or chronyd process is not running
                                                       execution time 0.01


[warning] node001: The trigger 'Failing health checks' is active because the measurable 'ntp' is FAIL (1)
[notice] basecm11: node001 [   UP   ], health check failed
```

Many scripts under `/cm/local/apps/cmd/scripts/` can have their debug output inspected with `samplenow --debug`.

A recursive grep on the head node, similar to the following, should show which scripts have a settable debug environment:

```
grep -r CMD_DEBUG /cm/local/apps/cmd/scripts/
```

The -s|--status Option

Nodes in device mode which have a status of UP, as seen by the status command, can be sampled with the -s|--status option:

Example

```
[basecm11->device]% samplenow -s UP
Entity    Measurable          Parameter  Type       Value             Age         State      Info
--------- ------------------- ---------- ---------- ----------------- ----------- ---------- ------
node001   AlertLevel          count      Internal   1                 1.59s                  ntp
node001   AlertLevel          maximum    Internal   15                1.59s                  ntp
node001   AlertLevel          sum        Internal   15                1.59s                  ntp
node001   BlockedProcesses               OS         0 processes       1.6s
node001   BufferMemory                   Memory     2.81 MiB          1.59s
node001   BytesRecv           ens3       Network    720.325 B/s       1.59s
node001   BytesSent           ens3       Network    934.768 B/s       1.59s
node001   CPUGuest                       CPU        0.00 Jiffies/s    1.6s
...
node002   AlertLevel          count      Internal   1                 1.04s                  ntp
node002   AlertLevel          maximum    Internal   15                1.04s                  ntp
node002   AlertLevel          sum        Internal   15                1.04s                  ntp
node002   BlockedProcesses               OS         0 processes       1.01s
node002   BufferMemory                   Memory     2.81 MiB          1.01s
node002   BytesRecv           ens3       Network    1.84 KiB/s        1.01s
node002   BytesSent           ens3       Network    2.05 KiB/s        1.01s
....
```

The preceding example is truncated because it is quite lengthy. However, on the screen, for the device mode, it shows all the sample values for the measurables for all the entities—head node and regular nodes—that are up.

To restrict the results to node001 only, it can be run as:

Example

```
[basecm11->device]% samplenow -s UP -n node001
Entity    Measurable          Parameter  Type       Value             Age         State      Info
--------- ------------------- ---------- ---------- ----------------- ----------- ---------- ------
node001   AlertLevel          count      Internal   1                 1.59s                  ntp
node001   AlertLevel          maximum    Internal   15                1.59s                  ntp
node001   AlertLevel          sum        Internal   15                1.59s                  ntp
...
```

Sampling according to a device status value other than UP is also possible.

The help text for the samplenow command gives further details on its possible options.

The latestmetricdata and latesthealthdata commands (section 10.6.3) display the results from the latest metric and health samples that have been gathered by the cluster, rather than sampling on demand.

The dumpmonitoringdata command (section 10.6.4) displays monitoring data gathered over a period of time in a variety of formats.

10.6.3 The Latest Data And Counter Values—The latest*data And latestmetriccounters Commands

Within device mode, the values obtained by the latest measurable sampling run can be displayed for a specified entity with the latestmonitoringdata, latestmetricdata and latesthealthdata commands:

- `latestmetricdata`: The `latestmetricdata` command for a specified entity displays the most recent metric value that has been obtained by the monitoring system for each metric used by the entity. For displaying metrics on-demand in `cmsh`, the `samplenow --metrics` command (page 629) can be used for a specified entity.
- `latesthealthdata`: The `latesthealthdata` command for a specified entity displays the most recent value that has been obtained by the monitoring system for each health check used by the entity. For displaying health check responses on demand in `cmsh`, the `samplenow --checks` command (page 629) can be used for a specified entity.
- `latestmonitoringdata`: The `latestmonitoringdata` command for a specified entity combines the output of the `latesthealthdata` and `latestmetricdata` commands, i.e. it displays the latest samples of the measurables for that entity. For displaying measurables on-demand in `cmsh`, the `samplenow` command (page 628) can be run without options, for a specified entity.

The `latestmetriccounters` command, on the other hand, displays the latest cumulative counter values of the cumulative metrics in use by the entity.

Using The `latest*data` Commands

When using the `latest*data` commands, the entity must be specified (some output elided):

Example

```
[basecm11->device]% use node001
[basecm11->device[node001]]% latestmetricdata
Measurable        Parameter  Type         Value              Age        State      Info
----------------- ---------- ------------ ------------------ ---------- ---------- ----------
AlertLevel        count      Internal     0                  1m 55s
AlertLevel        maximum    Internal     0                  1m 55s
AlertLevel        sum        Internal     0                  1m 55s
BlockedProcesses             OS           0 processes        2m 53s
BufferMemory                 Memory       1.96 MiB           3m 6s
BytesRecv         ens3       Network      137.199 B/s        2m 19s
BytesSent         ens3       Network      189.773 B/s        2m 19s
CPUGuest                     CPU          0.00 Jiffies/s     2m 53s
...
```

Valid entity grouping options and other options can be seen in the help text for the `latestmetricdata` and `latesthealthdata` commands.

Example

```
[basecm11->device]% help latestmetricdata
Name: Latestmetricdata - Display the latest metric data

Usage: latestmetricdata [OPTIONS] [<entity>]

Options:
    -v, --verbose
        Be more verbose

    -d, --delimiter <delimiter>
         Table delimiter

    -w, --width <size, [size, ...]>
         Overwrite table column width
```

```
    -m, --measurables <measurable,[measurable,...]>
         Comma separated list of measurables

    --class <class, [class, ...]>
         Show only the metrics of the specified classes

    -u, --unix-epoch
         Display unix epoch timestamps
...
```

The commands are mode-sensitive. That means, for example for a nodegroup consisting of, for example, `node001` and `node002`, that there is a difference in the entities that are displayed from `device` mode:

Example

```
[basecm11->device]% latestmetricdata -g mynodegroup
Entity       Measurable        Parameter  Type        Value            Age         State   Info
------------ ----------------- ---------- ----------- ---------------- ----------- ------- ------
node001      AlertLevel        count      Internal    0                52.5s
node001      AlertLevel        maximum    Internal    0                52.5s
node001      AlertLevel        sum        Internal    0                52.5s
node001      BlockedProcesses             OS          0 processes      1m 56s
node001      BufferMemory                 Memory      2.81 MiB         54.1s
node001      BytesRecv         ens3       Network     716.958 B/s      1m 14s
node001      BytesSent         ens3       Network     1.04 KiB/s       1m 14s
...
node002      AlertLevel        count      Internal    0                1m 37s
node002      AlertLevel        maximum    Internal    0                1m 37s
node002      AlertLevel        sum        Internal    0                1m 37s
node002      BlockedProcesses             OS          0 processes      2m 5s
node002      BufferMemory                 Memory      2.81 MiB         1m 3s
node002      BytesRecv         ens3       Network     2.26 KiB/s       1m 23s
node002      BytesSent         ens3       Network     2.80 KiB/s       1m 23s
...
```

and the entities that are displayed, for example, in `nodegroup` mode:

Example

```
[basecm11->nodegroup]% latestmetricdata mynodegroup
Measurable        Parameter    Type          Value      Age         State  Info
----------------- ------------ ------------ ---------- ----------- ------ -------
CoresDown                      Total        0          1m 18s
CoresTotal                     Total        4          1m 18s
CoresUp                        Total        4          1m 18s
DPUNodesClosed                 Total        0          1m 18s
DPUNodesDown                   Total        0          1m 18s
DPUNodesTotal                  Total        0          1m 18s
DPUNodesUp                     Total        0          1m 18s
DevicesWithLeaks               Total        0          1m 18s
FPGAsDown                      Total        0          1m 18s
FPGAsTotal                     Total        0          1m 18s
FPGAsUp                        Total        0          1m 18s
GPUsDown                       Total        0          1m 18s
GPUsTotal                      Total        0          1m 18s
```

```
GPUsUp                          Total       0          1m 18s
NodesClosed                     Total       0          1m 18s
NodesDown                       Total       0          1m 18s
NodesTotal                      Total       2          1m 18s
NodesUp                         Total       2          1m 18s
```

The metrics displayed in `device` mode are individual device metrics, while the metrics displayed in `nodegroup` mode are totalling metrics.

By default the data values are shown with human-friendly units. The `--raw` option displays the data values as raw units.

Using The `latestmetriccounter` Command

The `latestmetriccounter` is quite similar to the `latestmetricdata` command, except that it displays only cumulative metrics, and displays their accumulated counts since boot. The `latestmonitoringcounter` command is an alias for this command.

Example

```
[basecm11->device]% latestmonitoringcounters node001
Measurable         Parameter    Type         Value          Age         State  Info
------------------ ------------ ------------ -------------- ----------- ------ -------
BytesRecv          ens3         Network      558 MiB        2m 34s
BytesSent          ens3         Network      57 MiB         2m 34s
CPUGuest                        CPU          0 Jiffies      3m 15s
CPUIdle                         CPU          22.3 MJiffies  3m 15s
CPUIrq                          CPU          0 Jiffies      3m 15s
CPUNice                         CPU          148 Jiffies    3m 15s
...
```

The reader can compare the preceding example output against the example output of the `latestmetricdata` command (page 632) to become familiar with the meaning of cumulative output.

10.6.4 Data Values Over A Period—The `dumpmonitoringdata` Command

The `dumpmonitoringdata` command displays monitoring data values over a specified period. This is for an entity, such as:

- a node in `device` mode
- the `base` partition in `partition` mode
- an image in `softwareimage` mode
- a job in the `jobs` submode. The `jobs` submode is under the path `cmsh>wlm[<workload manager>]>jobs`, and using `dumpmonitoringdata` with it is covered on page 681.

Using The `dumpmonitoringdata` Command

A concise overview of the `dumpmonitoringdata` command can be displayed by typing in "`help dumpmonitoringdata`" in a `cmsh` mode that has entities.

The usage of the `dumpmonitoringdata` command consists of the following options and mandatory arguments:

`dumpmonitoringdata` [*OPTIONS*] <*start-time*> <*end-time*> <*measurable*> [*entity*]

The mandatory arguments: The mandatory arguments for the times, the measurables being dumped, and the entities being sampled, have values that are specified as follows:

- The measurable <*measurable*> for which the data values are being gathered must always be given. Measurables currently in use can conveniently be listed by running the `measurables` command (section 10.6.1).
- If [*entity*] is not specified when running the `dumpmonitoringdata` command, then it must be set by specifying the entity object from its parent mode of `cmsh` (for example, with `use node001` in `device` mode). If the mode is `device` mode, then the entity can also be specified via the options as a list, a group, an overlay, or a category of nodes.
- The time pair <*start-time*> or <*end-time*> can be specified as follows:
  - *Fixed time format*: The format for the times that make up the time pair can be:
    - [YY/MM/DD] HH:MM[:SS]
      (If YY/MM/DD is used, then each time must be enclosed in double quotes)
    - The unix epoch time (seconds since 00:00:00 1 January 1970)
  - `now`: For the <*end-time*>, a value of `now` can be set. The time at which the `dumpmonitoringdata` command is run is then used.
  - *Relative time format*: One item in the time pair can be set to a fixed time format. The other item in the time pair can then have its time set relative to the fixed time item. The format for the non-fixed time item (the relative time item) can then be specified as follows:
    - For the <*start-time*>, a number prefixed with "-" is used. It indicates a time that much earlier than the fixed end time.
    - For the <*end-time*>, a number prefixed with "+" is used. It indicates a time that much later than the fixed start time.
    - The number values also have suffix values indicating the units of time, as seconds (`s`), minutes (`m`), hours (`h`), or days (`d`).

    The relative time format is summarized in the following table:

Unit	<*start-time*>	<*end-time*>
seconds:	-<*number*>`s`	+<*number*>`s`
minutes:	-<*number*>`m`	+<*number*>`m`
hours:	-<*number*>`h`	+<*number*>`h`
days:	-<*number*>`d`	+<*number*>`d`

  - Both <*start-time*> and <*end-time*> can have their values prefixed with a "-". In this case, the range over which the monitored values are seen is in the past, relative to the current time. If the end time for the range is specified as further in the past than the starting time, then the time values are swapped over so that the end time becomes more recent than the starting time.

The options: The options applied to the samples are specified as follows:

Option	Argument(s)	Description
-v, --verbose		show the rest of the line on a new line instead of cutting it off
-d, --delimiter	"<*string*>"	set the delimiter to a character
-i, --intervals	<*number*>	number of samples to show
-i|--intervals <*number*> is mandatory if using one of the following four options:		
--sum		sum over specified entities
--max		maximum over specified entities
--min		minimum over specified entities
--avg		average over specified entities
-u, --unix-epoch		use a unix timestamp instead of using the default date format
--raw		show the raw value, without units
--human		show the human-friendly value, with appropriate units (default)
--consolidationinterval		only retrieve data from the consolidator associated with the specified consolidation interval. With default BCM values the consolidation interval used must be specified as 1 hour, which means it picks up the hourly consolidated data. A setting of zero means that the raw data values are presented.

--consolidationoffset		only retrieve data from the consolidator associated with that specified offset. The specified offset is the offset that is, for example, on the cmsh navigation path:
		
		`cmsh > monitoring > consolidator[default] > consolidators[hour] > set offset`
		
		or on the Base View navigation path:
		
		`Monitoring > Consolidators > default`  `> Consolidators`  `> hour` `> Offset`
		
		When specifying --consolidationoffset, the associated --consolidationinterval must also be specified.

...continues

...continued

Option	Argument(s)	Description
`--timeaverage`		calculate the average for the entire interval for specified devices
`--timesum`		calculate the sum for the entire interval for specified devices
`--timecount`		calculate the number of data points for the entire interval for specified devices
`--timemaximum`		calculate the maximum for the entire interval for specified devices
`--timeminimum`		calculate the minimum for the entire interval for specified devices
`--timegroup`		group data points for the entire interval for specified devices, intended for health checks and enummetrics
`--delta`		display change relative to previous value
`--clip`		clip data samples to the requested interval
`--uncompress`		uncompress data samples to the current sampling interval (shows intermediate values that are the same as the preceding value ("un-RLE" operation))
The following options are valid only for `device` mode:		
`-n, --nodes`	<*list*>	for list of nodes
`-g, --groups`	<*list*>	for list of groups
`-c, --categories`	<*list*>	for list of categories
`-r, --racks`	<*list*>	for list of racks
`-h, --chassis`	<*list*>	for list of chassis
`-e, --overlay`	<*list*>	Include all nodes in list of overlays
`--union`		calculate the union of specified devices
`--intersection`		calculate the intersection of the specified devices
`-l, --role`	<*role*>	Filter all nodes in role
`-s, --status`	<*state*>	for nodes in state UP, OPENING, DOWN, and so on

Notes And Examples Of `dumpmonitoringdata` Command Use

Notes and examples of how the `dumpmonitoringdata` command can be used now follow:

Fixed time formats: Time pairs can be specified for fixed times:

Example

```
[basecm11->device[node001]]% dumpmonitoringdata 23:34:00 23:38:00 loadone
# Start        - Sun Jan 18 23:34:00 2026 (1768808040)
# End          - Sun Jan 18 23:38:00 2026 (1768808280)
# LoadOne      - Load average on 1 minute
Timestamp                Value      Info
------------------------ ---------- ----------
2026/01/18 23:20:52.627    0
2026/01/18 23:36:52.627    0
2026/01/18 23:38:52.627    0.02
2026/01/18 23:40:52.628    0.02
```

Double quotes are needed for times with a YY/MM/DD specification:

Example

```
[basecm11->device[node001]]% dumpmonitoringdata "26/01/18 23:34" "26/01/18 23:38" loadone
# Start        - Sun Jan 18 23:34:00 2026 (1768808040)
# End          - Sun Jan 18 23:38:00 2026 (1768808280)
# LoadOne      - Load average on 1 minute
Timestamp                  Value      Info
-------------------------- ---------- ----------
2026/01/18 23:20:52.627    0
2026/01/18 23:36:52.627    0
2026/01/18 23:38:52.627    0.02
2026/01/18 23:40:52.628    0.02
```

Unix epoch time can also be set:

Example

```
[basecm11->device[node001]]% !date -d "Jan 18 23:34:00 2026" +%s
1768808040
[basecm11->device[node001]]% dumpmonitoringdata 1768808040 1768808280 loadone
# Start        - Sun Jan 18 23:34:00 2026 (1768808040)
# End          - Sun Jan 18 23:38:00 2026 (1768808280)
# LoadOne      - Load average on 1 minute
Timestamp                  Value      Info
-------------------------- ---------- ----------
2026/01/18 23:20:52.627    0
2026/01/18 23:36:52.627    0
2026/01/18 23:38:52.627    0.02
2026/01/18 23:40:52.628    0.02
```

If the start time is accidentally swapped with the end time, the time is intelligently swapped around by CMDaemon, with a warning.

Intervals and interpolation: The `-i|--intervals` option interpolates the data values that are to be displayed. The option needs <*number*> samples to be specified. This then becomes the number of interpolated samples across the given time range. Using "`-i 0`" outputs only the non-interpolated stored samples—the raw data—and is the default.

Example

```
[basecm11->device]% dumpmonitoringdata -i 0 -10m now loadone node001
# Start        - Sun Jan 18 23:43:41 2026 (1768808621)
# End          - Sun Jan 18 23:53:41 2026 (1768809221)
# LoadOne      - Load average on 1 minute
Timestamp                  Value      Info
-------------------------- ---------- ----------
2026/01/18 23:42:52.628    0
2026/01/18 23:44:52.627    0
2026/01/18 23:46:52.627    0.06
2026/01/18 23:48:52.628    0
2026/01/18 23:50:52.627    0.06
2026/01/18 23:52:52.627    0
```

If the number of intervals is set to a non-zero value, then the last value is always `no data`, since it cannot be interpolated.

Example

```
[basecm11->device]% dumpmonitoringdata -i 3 -10m now loadone node001
# Start        - Sun Jan 18 23:44:46 2026 (1768808686)
# End          - Sun Jan 18 23:54:46 2026 (1768809286)
# LoadOne      - Load average on 1 minute
Timestamp                  Value      Info
-------------------------- ---------- ----------
2026/01/18 23:44:46        0
2026/01/18 23:49:46        0.0266862
2026/01/18 23:54:46        no data
```

A set of nodes can be specified for the dump:

```
[basecm11->device]% dumpmonitoringdata -n node001..node002 -5m now cpuidle
# Start        - Sun Jan 18 23:50:33 2026 (1768809033)
# End          - Sun Jan 18 23:55:33 2026 (1768809333)
# CPUIdle      - CPU time spent in idle mode
Entity       Timestamp                  Value            Info
------------ -------------------------- ---------------- ----------
node001      2026/01/18 23:49:26.422    99.24 Jiffies/s
node001      2026/01/18 23:51:26.421    99.40 Jiffies/s
node001      2026/01/18 23:53:26.422    99.51 Jiffies/s
node002      2026/01/18 23:50:02.349    99.52 Jiffies/s
node002      2026/01/18 23:52:02.349    99.47 Jiffies/s
node002      2026/01/18 23:54:02.349    99.52 Jiffies/s
[basecm11->device]%
```

Summing values: The `--sum` option sums a specified metric for specified devices, for a set of specified times. For 2 nodes, over a period from 2 hours ago until now, with values interpolated over 3 time intervals, the option can be used as follows:

Example

```
[basecm11->device]% dumpmonitoringdata -2h now -i 3 loadone -n node00[1-2] --sum
# Start        - Sun Jan 18 21:58:48 2026 (1768802328)
# End          - Sun Jan 18 23:58:48 2026 (1768809528)
# LoadOne      - Load average on 1 minute
Timestamp                  Value      Info
-------------------------- ---------- ----------
2026/01/18 21:58:48        0.004537
2026/01/18 22:58:48        0.001543
2026/01/18 23:58:48        nan
```

Each entry in the value column in the preceding table is the sum of `loadone` displayed by `node001`, and by `node002`, at that time, as can be seen from the following corresponding table:

Example

```
[basecm11->device]% dumpmonitoringdata -2h now -i 3 loadone -n node00[1-2]
# Start        - Sun Jan 18 21:58:48 2026 (1768802328)
# End          - Sun Jan 18 23:58:48 2026 (1768809528)
# LoadOne      - Load average on 1 minute
Entity       Timestamp                  Value       Info
------------ -------------------------- ----------- ----------
node001      2026/01/18 21:58:48        0.001157
node001      2026/01/18 22:58:48        0.00154265
```

```
node001      2026/01/18 23:58:48        no data
node002      2026/01/18 21:58:48        0.00337953
node002      2026/01/18 22:58:48        0
node002      2026/01/18 23:58:48        no data
```

Each loadone value shown by a node at a time shown in the preceding table, is in turn an average interpolated value, based on actual data values sampled for that node around that time.

Maximum and minimum values: The --max option takes the maximum of a specified metric for specified devices, for a set of specified times. For 2 nodes, over a period from 2 hours ago until now, with values interpolated over 3 time intervals, the option can be run as follows:

Example

```
[basecm11->device]% dumpmonitoringdata -2h now -i 3 loadone -n node00[1-2] --max
# Start   - Tue Nov  3 09:56:05 2020 (1604393765)
# End     - Tue Nov  3 11:56:05 2020 (1604400965)
# LoadOne - Load average on 1 minute
Timestamp                  Value      Info
-------------------------- ---------- ----------
2020/11/03 09:56:05        0.010954
2020/11/03 10:56:05        0.000000
2020/11/03 11:56:05        nan
```

Each entry in the values column in the preceding table is the maximum of loadone displayed by node001, and by node002, at that time, as can be seen from the following corresponding table:

Example

```
[basecm11->device]% dumpmonitoringdata -2h now -i 3 loadone -n node00[1-2]
# Start   - Tue Nov  3 09:56:05 2020 (1604393765)
# End     - Tue Nov  3 11:56:05 2020 (1604400965)
# LoadOne - Load average on 1 minute
Entity       Timestamp                  Value      Info
------------ -------------------------- ---------- ----------
node001      2020/11/03 09:56:05        0.0109537
node001      2020/11/03 10:56:05        0
node001      2020/11/03 11:56:05        no data
node002      2020/11/03 09:56:05        0
node002      2020/11/03 10:56:05        0
node002      2020/11/03 11:56:05        no data
```

Similarly, for the preceding table, if the --min option is used instead, then the result would be:

Example

```
[basecm11->device]% dumpmonitoringdata -2h now -i 3 loadone -n node00[1-2] --min
# Start   - Tue Nov  3 09:56:05 2020 (1604393765)
# End     - Tue Nov  3 11:56:05 2020 (1604400965)
# LoadOne - Load average on 1 minute
Timestamp                  Value      Info
-------------------------- ---------- ----------
2020/11/03 09:56:05        0.000000
2020/11/03 10:56:05        0.000000
2020/11/03 11:56:05        nan
```

Displaying values during a specified time period, with `--clip`: The `--clip` option is used with a specified time period. If there are raw values within the period, these are displayed.

A value is displayed for the start of the period, either by selection of a raw value if it exists at the exact starting time, or via interpolation if there is no raw value at the exact starting time. Similarly, at the end of the period a raw value is shown if it exists, or an interpolated value is shown if it does not.

Example

```
dumpmonitoringdata -7m now BytesSent:ens3 --clip
# Start          - Mon Jan 19 00:13:20 2026 (1768810400)
# End            - Mon Jan 19 00:20:20 2026 (1768810820)
# BytesSent:ens3 - Sent bytes
Timestamp                  Value        Info
-------------------------- ------------ ----------
2026/01/19 00:13:20        195.054 B/s
2026/01/19 00:14:07        179.917 B/s
2026/01/19 00:16:07        285.856 B/s
2026/01/19 00:18:07        223.727 B/s
```

In the preceding example, the first sample is an interpolated value, interpolated over the raw samples at `00:12:07` and `00:14:07`. The remaining 3 are raw samples. The epoch times for this period, and corresponding human-readable values, are shown before the heading to the output table.

Displaying according to status: The `-s|--status` option selects only for nodes with the specified state. A state is one of the values output by the `cmsh` command `ds` or `device status`. It is also one of the values returned by the enummetric `DeviceStatus` (section 10.2.2).

Example

```
[basecm11->device]% dumpmonitoringdata  -2m now loadone -s up
# Start        - Mon Jan 19 00:25:28 2026 (1768811128)
# End          - Mon Jan 19 00:27:28 2026 (1768811248)
# LoadOne      - Load average on 1 minute
Entity     Timestamp                  Value      Info
---------- -------------------------- ---------- ----------
node001    2026/01/19 00:16:52.627    0
node001    2026/01/19 00:24:52.627    0
node001    2026/01/19 00:26:52.627    0.03
node002    2026/01/19 00:23:28.554    0.07
node002    2026/01/19 00:25:28.555    0
basecm11   2026/01/19 00:24:36.739    0.1
basecm11   2026/01/19 00:26:36.739    0.01
[basecm11->device]%
```

The argument to `-s|--status` can be specified with simple regexes, which are case insensitive. For example, `inst.*` covers the states `installing`, `installer_failed`, `installer_rebooting` `installer_calling_init`, `installer_unreachable`, `installer_burning`.

Displaying deltas: The `--delta` option lists the difference between successive monitoring data values. It subtracts the previous data value from the current data value, and divides the result by the time interval between the two values.

Example

```
[basecm11->device[node001]]% dumpmonitoringdata --delta -6m now pageout
# Start        - Mon Jan 19 00:23:54 2026 (1768811034)
# End          - Mon Jan 19 00:29:54 2026 (1768811394)
# PageOut      - Number of bytes the system has paged out to disk
Timestamp                  Value      Delta         Info
-------------------------- ---------- ------------ ----------
2026/01/19 00:22:32.863    7.1 KiB/s  nan
2026/01/19 00:24:32.864    8.7 KiB/s  13.8659 B/s/s
2026/01/19 00:26:32.863    5.1 KiB/s  -31.4304 B/s/s
2026/01/19 00:28:32.863    7.9 KiB/s  23.893 B/s/s
```

Deltas are useful for seeing patterns in rates of change. For example, to check an experimental version of CMDaemon for a memory leak, an administrator may run:

Example

```
[basecm11->device[basecm11]]% dumpmonitoringdata -8h now memoryused:cmd -n node001 --delta
# Start          - Sun Jan 18 16:59:27 2026 (1768784367)
# End            - Mon Jan 19 00:59:27 2026 (1768813167)
# MemoryUsed:cmd -
Timestamp                  Value      Delta        Info
-------------------------- ---------- ------------ ----------
2026/01/18 12:55:46.121    134 MiB    nan
2026/01/18 17:13:46.121    134 MiB    0 B/s
2026/01/18 17:15:46.121    136 MiB    21.3 KiB/s
2026/01/18 17:17:46.122    136 MiB    0 B/s
2026/01/18 17:19:46.122    135 MiB    -21.0 B/s
2026/01/19 00:57:46.121    134 MiB    0 B/s
```

The roughly 0B/s increase over 8 hours in the preceding output is a good sign.

Displaying union and intersection sets: The `--union` option displays the union of a set of specified devices. The devices can be specified by the device grouping options (the options that are used to group <*lists*>, such as `-c`, `-r` and so on).

For example:

if the overlay `galeranodes` has the node `mon001`

and

the overlay `openstackhypervisors` has the nodes `node001`, and `node002`

then an example of a union of the set of these two overlays is:

Example

```
[basecm11->device]% dumpmonitoringdata --union -3m now pageout -e galeranodes,openstackhypervisors
Entity       Timestamp                  Value       Info
------------ -------------------------- ----------- ----------
mon001       2018/09/11 11:31:56.198    192 KiB/s
mon001       2018/09/11 11:33:56.198    17.8 KiB/s
node001      2018/09/11 11:31:28.996    1.37 KiB/s
node001      2018/09/11 11:33:28.996    1.22 KiB/s
node002      2018/09/11 11:32:04.509    1.54 KiB/s
node002      2018/09/11 11:34:04.509    1.30 KiB/s
[basecm11->device]%
```

A union of sets in the same grouping option can be carried out using comma-separation for the list of sets. In the preceding example, the same grouping option is `-e|--overlay`.

For a union of different grouping options however, the syntax is different. For example, for a union of the `galeranodes` overlay, and a `node001` node, a similar example is:

Example

```
[basecm11->device]% dumpmonitoringdata --union -3m now pageout -u -e galeranodes -n node001
Entity       Timestamp                  Value        Info
------------ -------------------------- ------------ ----------
mon001       1536659036.198             17.3 KiB/s
mon001       1536659156.198             116 KiB/s
node001      1536659008.997             1023.99 B/s
node001      1536659128.996             1.26 KiB/s
[basecm11->device]%
```

For an intersection of sets, the only syntax allowed is one that uses different grouping options:

Example

```
[basecm11->device]% dumpmonitoringdata --intersection -3m now pageout -e galeranodes  -n node001
No remaining entities
```

For intersection, comma-separation within one grouping option is pointless, and is not supported.

Displaying percentages of a particular value across a time interval (the `--timegroup` option): The `--timegroup` option for a measurable displays the percentage of appearances of each sampled value of the measurable during the interval. The percentage is displayed in the row alongside the start time of the interval. The end time of the interval is displayed in the row that follows:

An example with `devicestatus` showing the percentages of times in the various provisioning states (section 5.5.3):

Example

```
[basecm11->device[node001]]% dumpmonitoringdata -8h now devicestatus --timegroup
# Start        - Mon May 13 07:43:13 2024 (1715578993)
# End          - Mon May 13 15:43:13 2024 (1715607793)
# DeviceStatus - The device status
Timestamp                  Value                  Info
-------------------------- ---------------------- ----------
2024/05/13 07:43:13        up                     72.8%
2024/05/13 15:43:13        up
2024/05/13 07:43:13        down                   24.1%
2024/05/13 15:43:13        down
2024/05/13 07:43:13        installing             1.46%
2024/05/13 15:43:13        installing
2024/05/13 07:43:13        installer_calling_init 0.31%
2024/05/13 15:43:13        installer_calling_init
2024/05/13 07:43:13        going_down             0.57%
2024/05/13 15:43:13        going_down
2024/05/13 07:43:13        booting                0.73%
2024/05/13 15:43:13        booting
```

Another example is with `wlm_slurm_state`, an enum that shows the state of nodes that can be allocated to the Slurm workload manager:

```
[basecm11->device[node001]]% dumpmonitoringdata -8h now wlm_slurm_state --timegroup
# Start           - Mon May 13 07:42:34 2024 (1715578954)
# End             - Mon May 13 15:42:34 2024 (1715607754)
# wlm_slurm_state - The state of the nodes
Timestamp                  Value      Info
-------------------------- ---------- ----------
```

```
2024/05/13 07:42:34        allocated  4.27%
2024/05/13 15:42:34        allocated
2024/05/13 07:42:34        drain      0.50%
2024/05/13 15:42:34        drain
2024/05/13 07:42:34        idle       90.9%
2024/05/13 15:42:34        idle
2024/05/13 07:42:34        maint      1.76%
2024/05/13 15:42:34        maint
2024/05/13 07:42:34        mixed      2.55%
2024/05/13 15:42:34        mixed
```

The percentage total is 100% in the output. The `--timegroup` option tends to be useful and meaningful for health checks and enummetrics, rather than for metrics.

Some non-interpolating RLE quirks: When a sample measurement is carried out, if the sample has the same value as the two preceding it in the records, then the "middle" sample is discarded from storage.

Thus, when viewing the sequence of output of non-interpolated samples, identical values do not exceed two entries one after the other. This is a common compression technique known as Run Length Encoding (RLE). It can have some implications in the output of the `dumpmonitoringdata` command.

Example

```
[basecm11->device[node001]]% dumpmonitoringdata -10m  now threadsused:cmd
# Start           - Mon Jan 19 01:15:01 2026 (1768814101)
# End             - Mon Jan 19 01:25:01 2026 (1768814701)
# ThreadsUsed:cmd -
Timestamp                  Value      Info
-------------------------- ---------- ----------
2026/01/19 01:13:30.233    42
2026/01/19 01:17:30.233    42
2026/01/19 01:19:30.234    44
2026/01/19 01:21:30.233    42
2026/01/19 01:23:30.233    44
```

In the preceding example, data values for the number of threads used by CMDaemon are dumped for the last 10 minutes.

Because of RLE, the value entry around `01:15:30` in the preceding example is skipped. It also means that at most only 2 of the same values are seen sequentially in the `Value` column. This means that 42 is not the answer to everything.

For a non-interpolated value, the nearest value in the past, relative to the time of sampling, is used as the sample value for the time of sampling. This means that for non-interpolated values, some care may need to be taken due to another aspect of the RLE behavior: The time over which the samples are presented may not be what a naive administrator may expect when specifying the time range. For example, if the administrator specifies a 10 minute time range as follows:

Example

```
[basecm11->softwareimage]% dumpmonitoringdata  -10m now nodesup default-image
# Start        - Mon Jan 19 01:21:25 2026 (1768814485)
# End          - Mon Jan 19 01:31:25 2026 (1768815085)
# NodesUp      - Number of nodes marked as UP
Timestamp                  Value      Info
-------------------------- ---------- ----------
2026/01/16 09:05:45.652    3
2026/01/19 01:29:45.651    3
```

then here, because the dump is for non-interpolated values, it means that the nearest value in the past, relative to the time of sampling, is used as the sample value. For values that are unlikely to change much, it means that rather than 10 minutes as the time period within which the samples are taken, the time period can be much longer. Here it turns out to be almost 3 days because the nodes happened to be booted then.

10.6.5 Monitoring Data Health Overview—The `healthoverview` Command

In figure 10.24, section 10.4.6, the Base View navigation path

```
Monitoring > Health Status
```

showed an overview of the health status of all nodes.

The `cmsh` equivalent is the `healthoverview` command, which is run from within `device` mode. If run without using a device, then it provides a summary of the alert levels for all nodes.

The help text in `cmsh` explains the options for the `healthoverview` command. The command can be run with options to restrict the display to specified nodes, and also to display according to the sort order of the alert level values.

Example

```
[basecm11->device]% healthoverview -n node00[1-3]
Device       Sum          Maximum      Count        Age          Info
------------ ------------ ------------ ------------ ------------ -------------
node001      30           15           2            50.7s        hot, fan high
node002      30           15           2            50.7s        hot, fan high
node003      15           15           1            50.7s        hot
```

10.6.6 Monitoring Data About The Monitoring Action History—The `monitoringactionhistory` Command

The `monitoringactionhistory` command displays a history of monitoring actions that have been triggered on an entity. This lets the cluster administrator examine which triggers fired on a node, and what actions were taken in response.

Example

```
[basecm11->device[node001]]% monitoringactionhistory
Entity      Measurable   Trigger                Action    Monitoring node    Timestamp
----------- ------------ ---------------------- --------- ------------------ -------------------
node001     ntp          Failing health checks  Event     basecm11           2026/02/27 12:16:21
node001     ntp          Passing health checks  Event     basecm11           2026/02/27 12:17:19
```

Each row shows the entity (the node here) that was triggered and executed the action, the measurable that was evaluated, the trigger that fired, the action that was executed, the monitoring node that processed it, and the timestamp.

The command is available in any entity mode (for example: `device`, `partition`, `wlm`), and can be used with the standard node selection options to filter by specific entities.

The recording of action history can be disabled by setting the `AdvancedConfig` directive `MonitoringEntityActionHistoryDisabled=1` (page 927). A restart of CMDaemon is required for this change to take effect.

10.6.7 Monitoring Data About The Monitoring System—The `monitoringinfo` Command

The `monitoringinfo` command provides information for specified head nodes or regular nodes about the monitoring subsystem. The help text shows the options for the command. Besides options to specify the nodes, there are options to specify what monitoring information aspect is shown, such as storage, cache, or services.

Example

```
[basecm11->device]% monitoringinfo -n node001
Service                         Queued     Handled    Cache miss Stopped    Suspended  Last operation
------------------------------- ---------- ---------- ---------- ---------- ---------- -------------------------
Mon::CacheGather                0          0          0          yes        no         -
Mon::DataConverter              0          0          0          yes        no         -
Mon::DataProcessorEngine        0          0          0          yes        no         -
Mon::DataProcessorKeepLatest    0          0          0          yes        no         -
Mon::DataProcessorKeepRecent    0          0          0          yes        no         -
Mon::DataProcessorMQTT          0          0          0          yes        no         -
Mon::DataProcessorPipe          0          0          0          yes        no         -
Mon::DataTranslator             0          267,192    20         no         no         Mon Jan 19 01:35:33 2026
Mon::EarlyWarning               0          0          20         no         no         Mon Jan 19 01:35:33 2026
Mon::EntityMeasurableCache      0          138        449        no         no         Fri Jan 16 08:55:22 2026
Mon::MeasurableBroker           0          46         56         no         no         Fri Jan 16 08:55:14 2026
Mon::OOB::TaskService           0          0          0          no         no         -
Mon::Offload::Balancer          0          0          0          yes        no         -
Mon::PerpetualTaskManager       0          0          0          no         no         -
Mon::RepositoryTrim             0          0          0          yes        no         -
Mon::TaskInitializer            0          28         0          no         no         Fri Jan 16 08:55:21 2026
Mon::TaskSampler                24         46,574     0          no         no         Mon Jan 19 01:35:33 2026
Mon::Trigger::Actuator          0          0          0          yes        no         -
Mon::Trigger::Dispatcher        0          0          0          yes        no         -
Mon::Trigger::Forward           0          0          0          yes        no         -
Prometheus::DataTranslator      0          0          0          yes        no         -

Cache                    Size          Updates      Requests     Lookup
------------------------ ------------ ------------ ------------ ------------
ConsolidatorCache        0             0            0            0
EntityCache              29            10           310,144      29
GlobalLastRawDataCache   0             10           0            0
LastRawDataCache         66            10           128,097      0
MeasurableCache          334           10           312,016      343

Cache                       Size          Added        Removed      Changes
--------------------------- ------------ ------------ ------------ ------------
LabeledEntityCache          0             0            0            0
VirtualLabeledEntityCache   0             0            0            0

Cache                Size          Skipped      Delayed      Nothing      Enriched
-------------------- ------------ ------------ ------------ ------------ ------------
JobEnrichService     0             0            0            0            0
JobExtraLabelCache   0             0            0            0            0

Cache              Up           Down         Closed       Muted
------------------ ------------ ------------ ------------ ------------
DeviceStateCache   4             0            0            0

Cache              Queued       Delivered    Handled      Pickup
------------------ ------------ ------------ ------------ ------------
Cache              41           138          267,172      1,941
PrometheusCache    0            0            0            1,941

Plotter             First      Last       Count    Samples  Sources  Requests
------------------- ---------- ---------- -------- -------- -------- --------
```

```
RequestDispatcher  -          -          0        0        0        -
RequestHandler     -          -          0        0        0        -

Storage                      Elements   Disk size    Usage    Free disk
---------------------------- ---------- ------------ -------- ------------
Mon::Storage::Engine         0          0 B          0.0%     -
Mon::Storage::Message        0          0 B          0.0%     -
Mon::Storage::RepositoryId   0          0 B          0.0%     -
[basecm11->device]%
```

10.6.8 Dropping Monitoring Data With The `monitoringdrop` Command

Monitoring data gathering can be restricted to certain nodes using node execution filtering and execution multiplexers. Entire data producers can also be disabled with the `disable` option in `monitoring` mode. However, restricting or disabling leaves historical samples in storage—the existing monitoring data values do not automatically get removed. So, in `cmsh` and Base View the latest known monitoring data values then still show up, with a forever-increasing age.

If a data producer is removed, then the associated data values for its measurable or measurables are removed.

Alternatively, if adding execution filters to a monitoring data producer is intended to be a permanent change, then all previously collected data can be dropped for filtered nodes.

For example, if the ssh connectivity to only cloud nodes is to be checked:

Example

```
[root@basecm11 ~]# cmsh
[basecm11]% monitoring setup use ssh2node
[...->monitoring->setup[ssh2node]]% executionmultiplexers
[...->executionmultiplexers]% show all nodes
Name                             All nodes
Revision
Type                             Type
Filter operation                 Include
Types                            Node
[...->executionmultiplexers]% use all nodes
[...->executionmultiplexers[All nodes]]% set types CloudNode
[...->executionmultiplexers*[All nodes*]]% commit
```

After this is set, the monitoring data values for a non-cloud node can be checked. The `ssh2node` health check data values are then seen to be getting older, without any more updates being added. These health check data values can then be dropped using the `monitoringdrop` command from within the `device` mode of `cmsh` command.

It is wise to run a dry-run operation first, in order to make sure that no data values are unintentionally removed:

Example

```
[root@basecm11 ~]# cmsh
[basecm11]% device use node001
[...->device[node001]]% latestmonitoringdata | grep ssh2node
ssh2node   Network      PASS       43m 38s     filtered
[...->device[node001]]% monitoringdrop --dry-run --filtered
Entity       Measurable
------------ ------------
node001      ssh2node
[...->device[node001]]% monitoringdrop --filtered
```

```
Removed 1 entity, measurable pairs
[...->device[node001]]% latestmonitoringdata | grep ssh2node
[...->device[node001]]%
```

The --force option can be used to remove non-filtered old data, such as data from a disabled measurable. This is also useful when correcting a bad metric script. After fixing the script, the old (incorrect) data can be dropped.

Example

```
[root@basecm11 ~]# cmsh
[basecm11]% device
[...->device]% monitoringdrop --category default my-metric --force
Removed 32 entity, measurable pairs
```

A reboot or CMDaemon restart is required for the node to start collecting data again on a non-filtered metric which has been dropped with the --force option.

10.6.9 Monitoring Suspension And Resumption—The monitoringsuspend And monitoringresume Commands

The monitoringsuspend command suspends monitoring. The monitoringresume command resumes monitoring.

When suspension is applied to a head node, the regular nodes simply continue sampling data up to a maximum of 1 million samples per node. The available backlog is fetched upon resumption.

Suspension can be used during benchmarking to measure the results of benchmarking runs without having monitoring get in the way.

Suspension can also be used as a quick sanity check during regular cluster operation, as a way for an administrator to see if it is monitoring that is consuming excessive resources, in comparison with the other processes on the system. For example, running it on a head node (some output omitted or elided):

Example

```
[root@head1 ~]# sar -b 1
Linux 3.10.0-957.1.3.el7.x86_64 (head1)  10/14/2019 _x86_64_ (28 CPU)

04:41:00 PM tps rtps wtps bread/s bwrtn/s
04:41:02 PM 1481.82 0.00 1481.82 0.00 24355.56
04:41:03 PM 849.49 0.00 849.49 0.00 10367.68
04:41:04 PM 509.00 0.00 509.00 0.00 4440.00
04:41:05 PM 709.90 0.00 709.90 0.00 5853.47
04:41:06 PM 1209.00 0.00 1209.00 0.00 18168.00
^C
[root@head1 ~]# cmsh
[head1]% device use master
[head1->device[head1]]% monitoringsuspend
suspend 14 on head1
[head1->device[head1]]% monitoringinfo
Service                      Queued     Handled    Cache miss Stopped    Suspended
---------------------------- ---------- ---------- ---------- ---------- ----------
Mon::CacheGather             425        39,857     0          no         yes
Mon::DataConverter           0          0          0          no         yes
Mon::DataProcessor           0          6,609,369  0          no         yes
Mon::DataTranslator          0          311,658    0          no         yes
Mon::EntityMeasurableCache   0          0          0          no         yes
Mon::MeasurableBroker        0          0          0          no         yes
```

```
Mon::PerpetualTaskManager   0          0          0          no         yes
Mon::Replicate::Collector   0          0          0          yes        yes
Mon::Replicate::Combiner    0          0          0          yes        yes
...
[head1->device[head1]]% quit
[root@head1 ~]# sar -b 1
Linux 3.10.0-957.1.3.el7.x86_64 (head1)  10/14/2019 _x86_64_ (28 CPU)

04:41:58 PM tps rtps wtps bread/s bwrtn/s
04:41:59 PM 4.04 0.00 4.04 0.00 96.97
04:42:00 PM 3.00 0.00 3.00 0.00 96.00
04:42:01 PM 4.04 0.00 4.04 0.00 96.97
04:42:02 PM 0.00 0.00 0.00 0.00 0.00
04:42:03 PM 0.00 0.00 0.00 0.00 0.00
04:42:04 PM 43.00 0.00 43.00 0.00 528.00
04:42:05 PM 0.00 0.00 0.00 0.00 0.00
04:42:06 PM 3.06 0.00 3.06 0.00 130.61
04:42:07 PM 0.00 0.00 0.00 0.00 0.00
```

In the preceding example monitoring is seen to be consuming significant resources.

After running `monitoringsuspend`, resuming monitoring should not be forgotten, and it should be done soon enough after suspension. If that is not done, then backlogged samples that exceed the limit of 1 million samples per node on the regular nodes would be lost. Resumption is carried out with:

Example

```
[root@head1 ~]# cmsh
[head1]% device use master
[head1->device[head1]]% monitoringresume
resume 14 on head1
```

CMDaemon Directive Settings To Reduce Monitoring Resource Consumption

The following CMDaemon directive changes may reduce the resource consumption due to monitoring:

Increasing the job account collection interval: by increasing the value of the `JobsSamplingMetricsInterval` directive (page 930).

Disabling job information collection completely: by setting the value of the `JobInformationDisabled` directive to `1` (page 934).

Reducing the duration for which job data is stored: by reducing the value of the `JobInformationKeepDuration` (page 935).

10.6.10 Monitoring Pickup Intervals

All nodes cache their monitoring data. This cached data gets picked up by the active head node at a regular pickup interval.

It is possible to alter the pickup interval using the `monitoringpickup` command covered in this section. The command is run from `device` mode.

The current pickup intervals can be listed with:

Example

```
[basecm11]% device
[basecm11->device]% monitoringpickup
Hostname      Interval    Times     Priority
```

```
------------ ---------- -------- ----------
basecm11     2m         -        0
node001      2m         -        0
node002      2m         -        0
```

An interval can be set for one or more nodes. For example, a 1-minute pickup interval can be set as follows:

Example

```
[basecm11]% device use node001
[basecm11->device[node001]]% monitoringpickup --interval 1m
Changed 1 pickup intervals
[basecm11->device[node001]]% monitoringpickup
Hostname     Interval    Times    Priority
------------ ----------- -------- ---------
node001      1m          1        100
```

The pickup interval is carried out only once by default, unless otherwise specified.

The `--times` option allows the number of times to be specified:

Example

```
[basecm11]% device use node001
[basecm11->device[node001]]% monitoringpickup --interval 1m --times 10
Changed 1 pickup intervals
[basecm11->device[node001]]% monitoringpickup
Hostname     Interval    Times    Priority
------------ ----------- -------- ---------
node001      1m          10       100
```

The `--forever` option lets the pickup be carried out forever.

Example

```
[basecm11->device[node001]]% monitoringpickup --interval 30s --forever
Changed 1 pickup intervals
[basecm11->device[node001]]% monitoringpickup
Hostname     Interval    Times    Priority
------------ ----------- -------- ---------
node001      30s         -        100
```

The `--priority` option applies the priority to equal or lower priority settings:

Example

```
[basecm11]% device
[basecm11->device]% monitoringpickup -n node00[1-2]
Hostname     Interval    Times    Priority
------------ ----------- -------- ---------
node001      1m          12       80
node002      1m          17       20
[basecm11->device]% monitoringpickup -n node00[1-2] --interval 5s --priority 50
Changed 1 pickup intervals
[basecm11->device]% monitoringpickup -n node00[1-2]
Hostname     Interval    Times    Priority
------------ ----------- -------- ---------
node001      1m          12       80
node002      5s          1        50
```

In the preceding example, parameters for node002 only were changed, as the priority setting for node001 was higher than the applied priority option that was requested. Thus, the Interval value became 5s, as specified, the Times value defaulted to 1, and the specified Priority value of 50 was applied to node002 only.

The further behavior of the pickup from node002 is as follows:

After picking up data once from node002, five seconds from the change, the interval becomes the default of 2 minutes once again:

```
[basecm11->device]% monitoringpickup -n node00[1-2]
Hostname      Interval    Times    Priority
------------- ----------- -------- ---------
node001       1m          12       80
node002       2m          0        0
```

The yet further behavior of the pickup, during the next pickup event, is then as follows:

The Times value of 0 becomes unset. The unset value is represented by -, and is equivalent to --forever.

In other words, if a monitoring interval is changed, and the change is not specified as "forever", then after the Times value has decremented to zero, the monitoring interval reverts to the default value of 2 minutes. The Times value then becomes a value of -, which implies forever, when the next pickup occurs.

The job metric sampler can also automatically modify the pickup interval for nodes. Every time a new job is started, all the nodes that are used by the job are assigned a modified pickup interval. The new values for the pickup can be managed in the jobmetricsettings mode of cmsh.

```
[basecm11->...->jobmetricsettings]% show
Parameter            Value
-------------------- ------
...
Pickup interval      5s
Pickup priority      50
Pickup times         12
```

10.7 Offloaded Monitoring

Traditional BCM monitoring uses a single (active) head node to manage monitoring. That is, to carry out sampling and to store results for measurables. Traditional monitoring can be used for clusters of thousands of nodes, assuming the default number of measurables are running.

Offloaded monitoring in BCM is designed to share the more resource-intensive parts of monitoring across nodes so that the head node is not overloaded by monitoring. In practice, offloaded monitoring needs only to be considered for a clusters that are greater than about 1000 nodes in size, assuming the clusters have the default number of measurables running.

There are some mandatory requirements, and some recommended settings, which are discussed later on in section 10.7.3.

10.7.1 Why Offloaded Monitoring?

Traditional monitoring is highly optimized, and with some care is typically able to deal with clusters of around 10,000 nodes with the default metrics. While it has the virtue of simplicity, it also has the following possible issues:

- there is a single point of failure, since monitoring runs on the active head node
- the head node performance as the number of nodes increases may not be sufficient. To get around this, monitoring may rely on increasingly expensive hardware, or on reducing the sampling that

is carried out. With the default monitoring in place, with typical server hardware available at the time of writing of this section (2020), a limit is reached at around 20000 nodes.

These issues may not be acceptable, in which case it makes sense to consider offloaded monitoring. The advantages of offloaded monitoring are:

- no single point of failure
- the ability to scale with the size of the cluster

A disadvantage is that offloaded monitoring is more complicated than single head monitoring. However, BCM simply implements it as a role that is assigned to nodes. The BCM backend then manages the details of offloaded monitoring.

10.7.2 Implementing Offloaded Monitoring

In `cmsh` offloaded monitoring is implemented via role assignment. The assignment can be carried out at the level of device, category, or configuration overlay:

Example

```
[basecm11->device]% use node001
[basecm11->device[node001]]% roles
[basecm11->device[node001]->roles]% assign monitoring
[basecm11->device*[node001*]->roles*[monitoring*]]% show
Parameter                      Value
------------------------------ ------------------------------------------------
Name                           monitoring
Revision
Type                           MonitoringRole
Add services                   yes
Number of backups              2
Backup ring                    automatic
Maximum number of nodes        16384
Delay after up                 1m
Delay after down               10m
Backup on shutdown             no
Backup on reboot               no
Backup on power off            no
Backup on power reset          no
Backup on terminate            no
Node filters                   <0 in submode>
[basecm11->device*[node001*]->roles*[monitoring*]]%
```

If offloaded monitoring is to run in a highly-available way, so that a failure of one monitoring node does not halt the monitoring system, then offloaded monitoring must be assigned to two or more nodes.

10.7.3 Background Details

A description of how offloaded monitoring works in the backend follows, because it should help the cluster administrator in understanding how and when to implement it.

Offloaded monitoring uses nodes that are assigned a monitoring role.

If there are N regular (non-head) nodes in a cluster that are being monitored, and if there are M monitoring nodes, then the idea of offloading is that each monitoring node covers N/M of the total monitoring storage, and N/M of the sampling scripts.

In other words, the cluster manager aims to evenly spread the total storage and sampling needed for all the regular nodes, over the nodes with a monitoring role.

BCM in the default state with no high availability does not run offloaded monitoring.

High Availability And Offloaded Monitoring With Just The Head Nodes Running As Monitoring Nodes

The simplest offloaded monitoring configuration is when high availability is configured. That is, when BCM is configured with two head nodes as described in Chapter 15. By default, a monitoring role is then assigned to both the head nodes.

This has the effect of doubling the monitoring capacity of the head node pair in NVIDIA Base Command Manager 9.1, in comparison with a head node pair in NVIDIA Base Command Manager version 9.0 and earlier.

The head nodes then carry out storage and sampling for the regular nodes as well as for themselves.

Offloaded Monitoring With Regular Compute Nodes Running As Monitoring Nodes

It is possible to run a compute node with a monitoring role assigned to them. This means that the compute node carries out storage and sampling as part of its monitoring role.

During a SYNC install—the default node provisioning for a healthy node—monitoring data persists.

Monitoring data would be wiped out during a FULL install (section 5.4.4). To provide a check on this, the node can be set up with the `datanode` setting (page 279), which requires a confirmation from the cluster administrator before carrying out a FULL install. However, if the monitoring data values are that important, then the cluster administrator should consider backup solutions for it anyway.

Offloaded Monitoring With Dedicated Nodes Running As Monitoring Nodes

For large clusters of around 10,000 or more nodes, a recommended practice is to have dedicated monitoring nodes. These are then regular nodes that are typically set up with the `datanode` setting, and are not used for other purposes such as HPC use. The dedicated monitoring nodes then carry out monitoring sampling and monitoring data storage for the regular nodes. Each of the M dedicated monitoring nodes takes on N/M of the regular nodes for itself, and records monitoring data from those N/M nodes.

This is illustrated by the following schematic, with arrows indicating the monitoring sampling flow for the head nodes (H1, H2), dedicated monitoring nodes (M1 to M3), and regular nodes (N1 to N6):

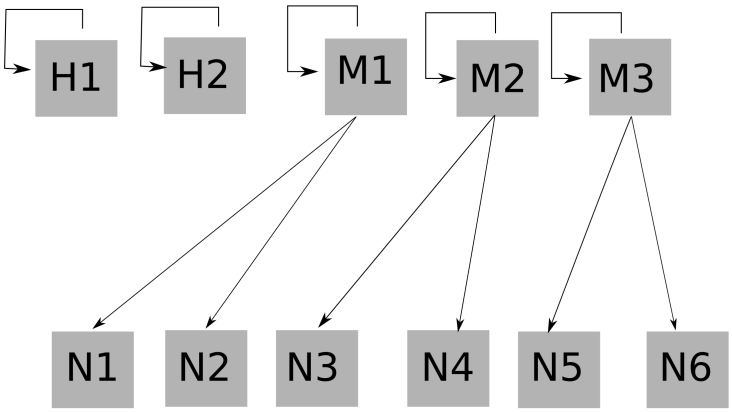


Figure 10.28: Monitoring Sampling Flow For Offloaded Monitoring With Dedicated Monitoring Nodes

A monitoring node in this configuration also copies backups of its monitoring data to other monitoring nodes. `Number of backups` for the monitoring role (section 10.7.2) is used to configure the number of backups. In the following schematic, two neighboring monitoring nodes are used as backup:

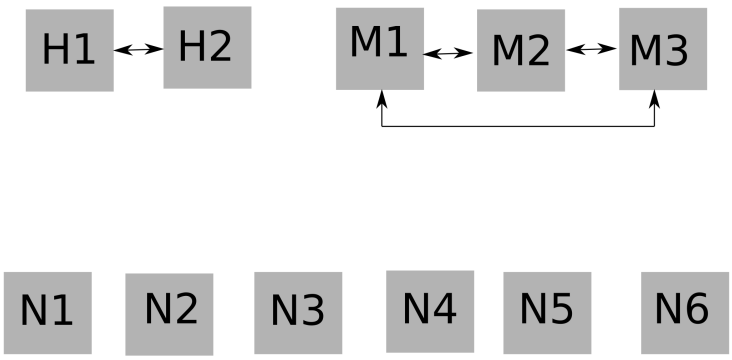


Figure 10.29: A Simple Backup Flow For Offloaded Monitoring With Dedicated Monitoring Nodes

The backups need not be on the same local network. For example, edge directors can be backed up to the head node.

If a monitoring node fails, then its monitoring data can be extracted from its backup nodes, and a new distribution of nodes to be monitored is allocated to the remaining monitoring nodes.

A backup is carried out using what the BCM developers call a *provisioning grab*. This is similar to grabimage (section 5.6), but this time designed for grabbing monitoring data. Like grabimage, provisioning grab also works on the basis of an rsync. This means that the first copy can take a while, but that subsequent copies are much faster.

Provisioning grabs are staggered to reduce bandwidth consumption and to reduce the likely amount of monitoring data that goes out of date during an outage.

Dedicated monitoring nodes can cope with short outages of monitoring nodes, such as are caused by a CMDaemon restart on that monitoring node, or by a reboot of that monitoring node. These outages are not expected to take longer than a few minutes, and the monitoring nodes just continue on as normal, with some missing data samples. However, if an outage is greater than about 15 minutes, such as may happen if a monitoring node crashes, then a fully automated rebalancing of the loads on the monitoring nodes can only take place with the aid of backups.

The head nodes in this configuration are configured as HA, and without the monitoring role, and thus do not carry out monitoring data storage for the regular nodes. They do however still sample and store data for themselves, and carry out backups to each other.

Backup nodes: In addition, for larger clusters, another recommended practice is to have backup nodes (B1, B2 in the following schematic) for the dedicated monitoring nodes:

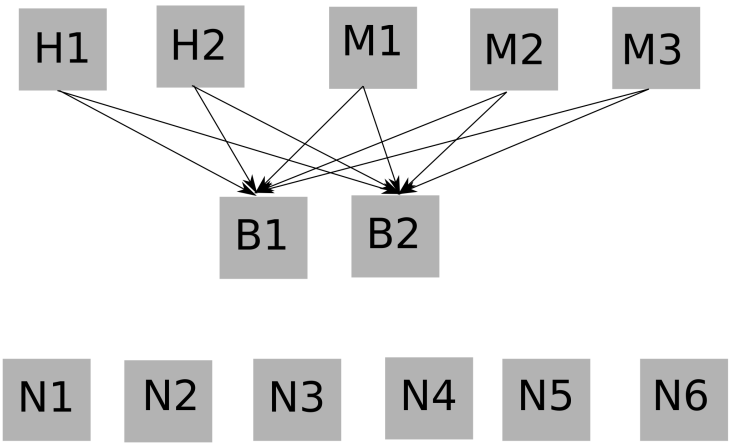


Figure 10.30: A More Sophisticated Backup Flow For Offloaded Monitoring With Dedicated Monitoring Nodes And Dedicated Backup Nodes

Backup nodes for the monitoring nodes take away the monitoring data backup task from the monitoring nodes. This frees up the monitoring nodes so that they can take on even more monitoring.

Provisioning role on monitoring nodes: If there is enough capacity on the dedicated monitoring nodes, and the cluster spends most of its time in a relatively steady state where its nodes do not reboot frequently, then adding a provisioning role to the monitoring nodes can be an efficient use of resources. In this case the monitoring nodes are obviously not so dedicated, but the advantage is that rebooting the entire cluster is then faster, at the cost of perhaps some extra load on the monitoring nodes during such a reboot.

Offloaded Monitoring Sampling And Backup Flows For Edge Computing

For a cluster with edge configured, the edge director flows in the edge network are analogous to head node flows in the local network. Thus, monitoring is carried out by the directors on the edge nodes, and the directors also sample themselves.

Thus, edge directors, not in a high-availability configuration, have the monitoring sampling data flow shown by the following schematic (figure 10.31):

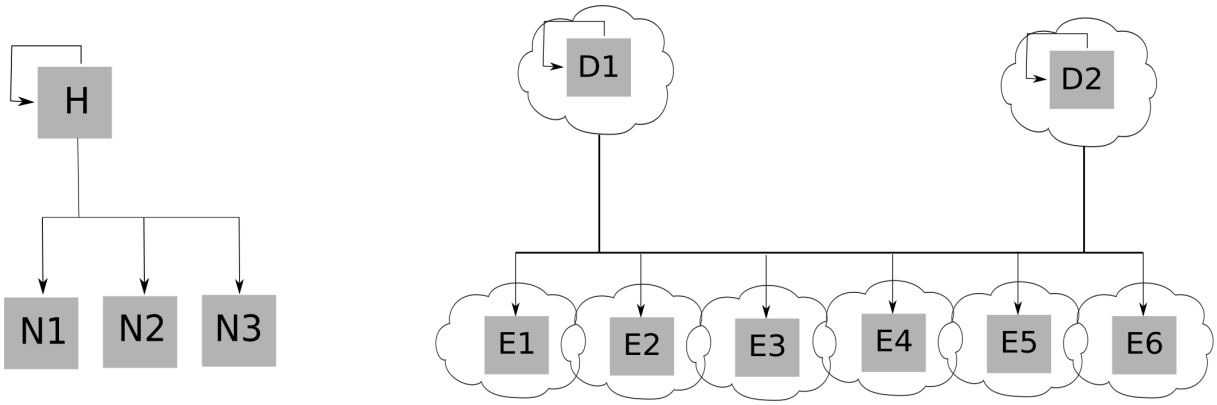


Figure 10.31: Sampling Flow For Offloaded Monitoring With Non-HA Edge Director Nodes

For edge directors that have been set up in an HA configuration (section 2.1.1 of the *Edge Manual*) the monitoring sampling data flow in the edge network is split up between directors, so that each director takes half of the edge nodes. This is analogous to how head nodes in an HA configuration take half of the regular nodes each (figure 10.32):

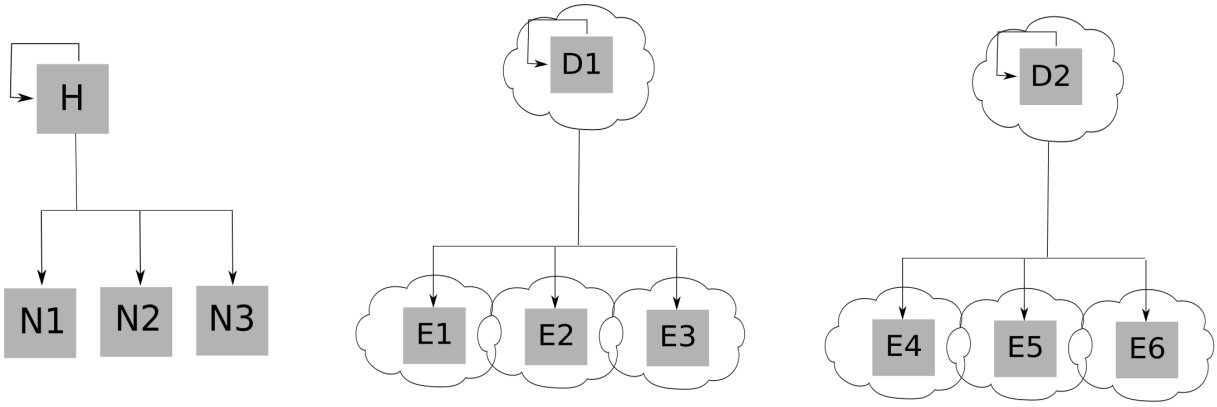


Figure 10.32: Sampling Flow For Offloaded Monitoring With HA Edge Director Nodes

The backup data flow for a non-HA configuration would then be as follows for an edge director (figure 10.33):

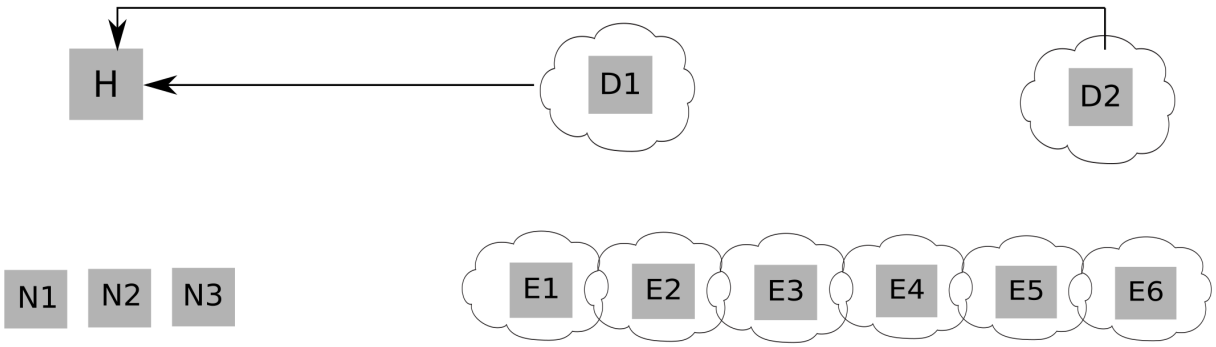


Figure 10.33: Backup Flow For Offloaded Monitoring With Non-HA Edge Director Nodes

Backing up to the head node is possible for an edge director. But it is usually unwise because one of the usual reasons to have a segregation of local and edge networks is to reduce data flow between the local and edge network.

With edge directors in an HA configuration, a big advantage is that backing up to the other edge director is possible and configured by default, rather than backing up to the head node (figure 10.34):

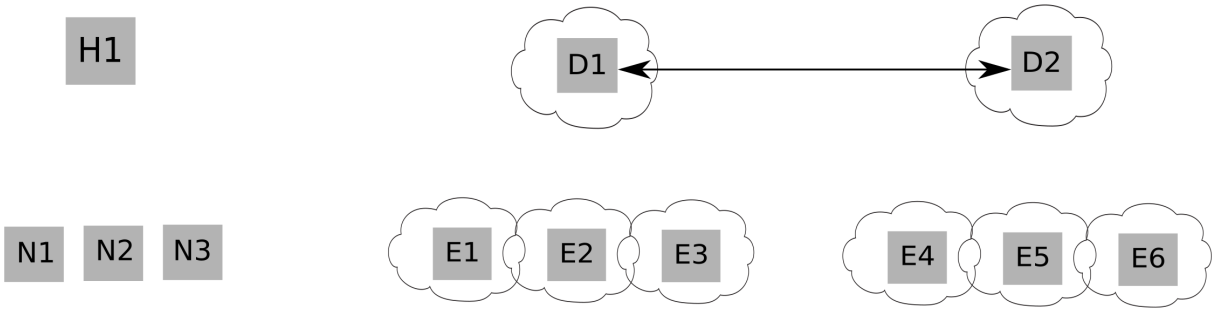


Figure 10.34: Backup Flow For Offloaded Monitoring With HA Edge Director Nodes

Default Backups Configurations

The default backup configurations for monitoring data are:

- Head node HA : head nodes back up each other
- Edge node HA : directors back up each other
- edge directors: directors back up to (both) head nodes
- cloud directors: directors back up to (both) head nodes

10.7.4 Examining Offloaded Monitoring With `monitoringoffloadinformation`

The `monitoringoffloadinformation` displays the monitoring relations between nodes. In a small HA cluster with a default configuration, with only two HA head nodes `basecm11-ha-a` and `basecm11-ha-b` in a monitoring role, the output of the command for `node001` is:

```
[basecm11-ha-a->device[node001]]% monitoringoffloadinformation
Node     Selected Monitoring node   Viable Monitoring nodes
-------- -------------------------- --------------------------
node001  basecm11-ha-b              basecm11-ha-a,basecm11-ha-b
```

Here, `node001` is seen as having its monitoring data going to one selected head node.

Viable in this context means a node that is capable of being used for monitoring, even if it may not be available now, for example due to a temporary outage such as a reboot. Both head nodes are thus capable of doing monitoring.

For one of the head nodes in the cluster, the output is:

```
[basecm11-ha-a->device[basecm11-ha-a]]% monitoringoffloadinformation
Node          Selected Monitoring node   Viable Monitoring nodes
------------- -------------------------- --------------------------
basecm11-ha-a basecm11-ha-a              basecm11-ha-a
```

10.8 The User Portal

The user portal is a restricted version of Base View that allows non-root users to view some cluster manager data.

With a browser:

- If the head node landing page (figure 2.1) shows a greytoned user portal block with a ⊕ within it, then it means that the user portal is not installed.
- If the head node landing page shows a colored user portal block with a chain link icon within it, then the user portal can be accessed via the icon.

The user portal can be added or removed from the cluster manager by adding or removing the `cm-webportal` package.

Example

```
[root@basecm11 ~]# yum install cm-webportal
...
Is this ok [y/N]: y
Downloading Packages:
...
Complete!
```

10.8.1 Accessing The User Portal

The user portal is compatible with most browsers using reasonable settings, and is supported for the same browsers that Base View supports (section 2.4).

The user portal is located by default on the head node, and can then be accessed in two ways:

- From the chain link icon within the colored user portal block of the head node landing page, mentioned earlier.
- More directly using a URL of the form:

  `https://`*<host name or IP address>*`:8081/userportal`

Both of these access routes lead to a user login page. The state of the cluster can then be viewed by the users via an interactive interface.

The first time a browser is used to log in to the portal, a prominent warning about the site certificate being untrusted appears.

This is because certificate is a self-signed certificate (the X509v3 certificate of Chapter 4 of the *Installation Manual*), generated and signed by BCM, and the attributes of the cluster owner are part of the certificate. However, the CA for BCM (`NVIDIA Bright CA`) is not a recognized Certificate Authority (CA) like the CAs that are recognized by a browser, which is why the warning appears.

For a portal that is not accessible from the outside world, such as the internet, the warning about Bright Computing not being a recognized Certificate Authority is not an issue, and the user can simply accept the "untrusted" certificate, and the browser used then no longer displays such a prominent warning about the issue.

For a portal that is accessible via the internet, some administrators may regard it as more secure to ask users to trust the self-signed certificate rather than external certificate authorities. Alternatively the administrator can replace the self-signed certificate with one obtained by a trusted recognized CA, for example the one at `https://letsencrypt.org`, if that is preferred.

The user portal certificate discussed here is a webserver certificate, similar to that of the landing page, but served by CMDaemon rather than Apache.

10.8.2 Setting A Common Username/Password For The User Portal

By default, each user has their own username/password login to the portal. Removing the login is not possible, because the portal is provided by CMDaemon, and users must connect to CMDaemon.

A shared (common) username/password for all users can be set in the configuration file, `common-credentials.json`. The default username/password settings are blank, which means that common access is not enabled:

Example

```
[root@basecm11 ~]# cat /cm/local/apps/cmd/etc/htdocs/userportal/assets/config/common-credentials.json
{
  "username": "",
  "password": ""
}
```

To enable common access:

- the common username and password must be added via `cmsh` or Base View

  Example

  ```
  [root@basecm11 ~]# cmsh
  [basecm11]% user
  [basecm11->user]% add forrestgump
  [basecm11->user[forrestgump]]% set password
  enter new password:
  retype new password:
  [basecm11->user*[forrestgump*]]% commit
  [basecm11->user[forrestgump]]% quit
  ```

- the common username and password should be set in the appropriate place in the configuration file, `common-credentials.json`:

  Example

  ```
  [root@basecm11 ~]# cat /cm/local/apps/cmd/etc/htdocs/userportal/assets/config/common-credentials.json
  {
    "username": "forrestgump",
    "password": "1forrest1"
  }
  ```

A minor stumbling block for the unwary administrator is:

If using Base View, then if the password for the username has already been saved in the browser's password manager before changing it in the configuration file, then the password saved in the browser's password manager may need to be changed to the new one explicitly.

10.8.3 User Portal Access

By default, the user profile (section 6.5) is set to `readonly`, which allows viewing of the information presented in the user portal, without allowing it to be altered.

10.8.4 User Portal Home Page

User Portal Overview Page

The default user portal home page is the `Overview` page. This allows a quick glance to convey the most important cluster-related information for users (figure 10.35):

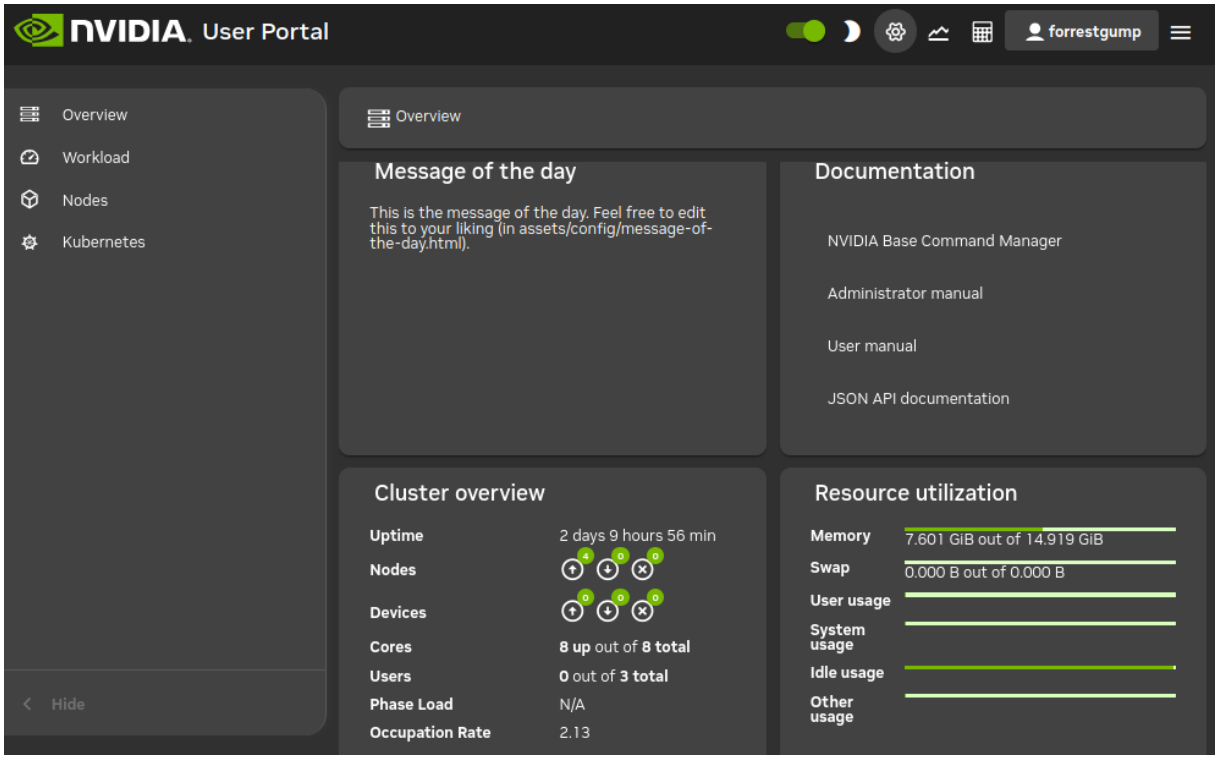


Figure 10.35: User Portal: Overview Page

The following items are displayed on the overview page:

- a Message Of The Day. This can be edited in `/cm/local/apps/cmd/etc/htdocs/userportal/assets/config/message-of-the-day.html`
- links to the documentation for the cluster
- an overview of the cluster state, displaying some cluster parameters. By default, it is refreshed every 10s.

The user portal is designed to serve files only, and will not run executables such as PHP or similar CGI scripts.

User Portal Job Accounting Page

Job accounting charts can be viewed on clicking upon the associated icon, , at the top right corner of the user portal page. The user portal's `Accounting and reporting` page for Base View is then displayed.

The accounting and reporting page allows job accounting to be viewed in an accounting panel in a very similar manner to how it is done in section 12.5.

10.9 Cloud Job Tagging

Cloud job tagging is about the ability for cloud job instances to have their associated cloud resources *tagged*. This is only possible for AWS at the time of writing (February 2020). Enabling cloud job tagging via NVIDIA Base Command Manager was introduced in version 9.0.

Tags are key=value pairs for AWS resources, and can be *applied* to resources. Typically, tags that are applied are set by the user via the Tag Editor of the Amazon Management Console, and up to 50 tags can be applied per resource.

Cloud job tagging should not be confused with the tagging of job metrics for job accounting (section 12.2). AWS cloud resource tagging is only active and handled within AWS.

Cloud job tags allow the time span between tag creation and removal to be associated with a particular workload on the node.

In `cmsh`, for a cloud node, cloud job tagging can be enabled within `cloud` mode by setting the `cloudjobtagging` parameter for the EC2Provider entity to `yes`

Example

```
cmsh -c 'cloud; use amazon; set cloudjobtagging yes; commit'
```

If it is set to `yes`, then every job running on a cloud node using that specific provider is tagged according to the applied tags.

A subset of the tags for cloud jobs are *cost allocation tags*. Cloud job cost allocation tags allow AWS costs to be tracked for jobs. A cost allocation tag can be:

- an AWS generated tag: defined, created, and applied by AWS
- a user-defined tag: defined, created, and applied by the user

By default, BCM provides the following tag names when the cloud job tagging feature is enabled:

- `BCM_JOB_ID`
- `BCM_JOB_ACCOUNT`
- `BCM_JOB_USER`
- `BCM_JOB_NAME`

When CMDaemon sees that a job has started, the resources of that job are then tagged with the job ID, the job account, the job user, and the job name. When CMDaemon detects that the job has stopped, it removes the tags.

The AWS Cost Explorer can be used to view the AWS costs for a billing period according to tags.

Further information on tagging can be found at:
`https://docs.aws.amazon.com/AWSEC2/latest/UserGuide/Using_Tags.html`.

Further information on using the Cost Explorer with cost allocation tags can be found at:
`https://docs.aws.amazon.com/awsaccountbilling/latest/aboutv2/cost-alloc-tags.html`

10.10 Event Viewer

Monitoring in BCM is normally taken by developers to mean how sampling with data producers is handled. However, cluster administrators using this manual typically consider watching and handling events in BCM to also be a part of a more general concept of monitoring. This manual is aimed at cluster administrators, and therefore this section on event viewing and handling is also placed in the current monitoring chapter.

BCM events can be handled and viewed in several ways. Event logging is enabled by default by the `EventLogger` directive (page 916).

10.10.1 Viewing Events In Base View

In Base View, events can be viewed by clicking on the `Events` icon of figure 10.5. This opens up a window with a sortable set of columns listing the events in the events log, and by default with the most recent events showing up first.

10.10.2 Viewing Events In cmsh

The events command is a global cmsh command. It allows events to be viewed at several severity levels (section 10.2.7), and allows old events to be displayed. The usage and synopsis of the events command is:

```
Usage:    events
          events on [broadcast|private]
          events off [broadcast|private]
          events show-acknowledged <yes/no>
          events show-detailed <yes/no>
          events level <level>
          events clear
          events details <id> [<id>]
          events acknowledge <id> [<id>]
          events unacknowledge <id> [<id>]
          events <number> [level]
          events follow

Arguments:
          level     info, notice, warning, error, alert
```

Running the command without any option shows event settings:

Example

```
[basecm11->device]% events
Private events:      off
Broadcast events:    on
Acknowledged events: off
Details:             off
Level:               notice
```

Running the command with options allows the viewing and setting of events as follows:

- on [broadcast|private]: event messages are displayed as they happen in a session, with cmsh prompts showing in between messages:
  - If only on is set, then all event messages are displayed as they happen:
    - either to all open cmsh sessions, and also in Base View event viewer panes, if the event or its trigger has the "broadcast" property.
    - or only in the cmsh session that is running the command, if the event or its trigger has the "private" property.
  - If the further option broadcast is set, then the event message is displayed as it happens in all open cmsh sessions, and also in all Base View event viewer panes, if the event or its trigger has the "broadcast" property.
  - If the further option private is set, then the event message is displayed as it happens only in the cmsh session that ran the command, if the event or its trigger has the "private" property.
- off [broadcast|private]: disallows viewing of event messages as they happen in a session. Event messages that have not been displayed due to being forbidden with these options, are displayed when the events command is run without any options in the same session.
  - If only off is set, then no event message is displayed as it happens in the session. This is regardless of the "broadcast" or "private" property of the event or its trigger.

  - If the further option `broadcast` is set, then the event message is not displayed as it happens, if the event or its trigger has the "broadcast" property.
  - If the further option `private` is set, then the event message is not displayed as it happens, if the event or its trigger has the "private" property.

- `show-acknowledged`: if set to `yes`, then `events` shows the events that have been acknowledged
- `show-detailed`: if set to `yes`, then `events` shows the events in greater detail, if possible
- `level <info|notice|warning|error|alert>`: sets a level. Messages are then displayed for this and higher levels.
- `clear`: clears the local `cmsh` event message cache. The cache indexes some of the events.
- `details` <*id*>: shows details for the event with the index value of <*id*>, which is a number that refers to an event.
- `acknowledge` <*id*>: acknowledges the event with the index value of <*id*>
- `unacknowledge` <*id*>: unacknowledges the event
- <*number*> `[info|notice|warning|error|alert]`: shows a specified <*number*> of past lines of events. If an optional level (`info`, `notice`,...) is also specified, then only that level and higher (more urgent) levels are displayed.
- `follow`: follows event messages in a `cmsh` session, similar to `tail -f /var/log/messages`. This is useful, for example, in tracking a series of events in a session without having the `cmsh` prompt showing. The output can also be filtered with the standard unix text utilities, for example: `events follow | grep node001`

A common example of events that send private messages as they happen are events triggered by the `updateprovisioners` command, which has the "private" property. The following example illustrates how setting the event viewing option to `private` controls what is sent to the `cmsh` session. Some of the output has been elided or truncated for clarity:

Example

```
[basecm11->softwareimage]% events on private
Private events:   on
[basecm11->softwareimage]% updateprovisioners
Provisioning nodes will be updated in the background.
[basecm11->softwareimage]%
Tue Apr 29 01:19:12 2014 [notice] basecm11: Provisioning started: sendi...
[basecm11->softwareimage]%
Tue Apr 29 01:19:52 2014 [notice] basecm11: Provisioning completed: sen...
updateprovisioners [ COMPLETED ]
[basecm11->softwareimage]% !#events were indeed seen in cmsh session
[basecm11->softwareimage]% !#now block the events and rerun update:
[basecm11->softwareimage]% events off private
Private events:   off
[basecm11->softwareimage]% updateprovisioners
Provisioning nodes will be updated in the background.
[basecm11->softwareimage]% !#let this 2nd update run for a while
[basecm11->softwareimage]% !#(time passes)
[basecm11->softwareimage]% !#nothing seen in cmsh session.
[basecm11->softwareimage]% !#show a 2nd update did happen:
[basecm11->softwareimage]% events 4 | grep -i provisioning
```

```
Tue Apr 29 01:19:12 2014 [notice] basecm11: Provisioning started: sendi...
Tue Apr 29 01:19:52 2014 [notice] basecm11: Provisioning completed: sen...
Tue Apr 29 01:25:37 2014 [notice] basecm11: Provisioning started: sendi...
Tue Apr 29 01:26:01 2014 [notice] basecm11: Provisioning completed: sen...
```

10.10.3 Using The Event Bucket From The Shell For Events And For Tagging Device States

Event Bucket Default Behavior

The BCM *event bucket* accepts input piped to it, somewhat like the traditional unix "bit bucket", `/dev/null`. However, while the bit bucket simply accepts any input and discards it, the event bucket accepts a line of text and makes an event of it. Since the event bucket is essentially an event processing tool, the volumes that are processed by it are obviously less than that which `/dev/null` can handle.

By default, the location of the event bucket is at `/var/spool/cmd/eventbucket`. Directing a message to the event bucket makes an event, which can be viewed in `cmsh` or Base View.

Example

```
[root@basecm11 ~]# echo "Some text" > /var/spool/cmd/eventbucket
```

This adds an event with, by default, the `info` severity level. It can be seen within `cmsh` with the `events` command:

Example

```
[basecm11]% events 3
Thu Jan 22 10:02:59 2026 [notice] basecm11: Slurm topology updated (13)
Thu Jan 22 10:02:59 2026 [notice] basecm11: Service slurmctld was restarted
Thu Jan 22 13:17:22 2026 [notice] basecm11: Some text (11)
```

or it can be viewed in the Base View event pane that is accessible with the event icon in the top right corner of the Base View standard display (figure 10.5).

10.10.4 InfoMessages

InfoMessages are optional messages that inform the administrator of the reason for the status change of a measurable, or an event in the cluster.

Measurable scripts can use file descriptor 3 within their scripts to write an InfoMessage:

Example

```
echo "Drive speed unknown: Reverse polarity" >&3
```

Event Bucket Severity Levels

To write events at specific severity levels (section 10.2.7), and not just at the `info` level, the appropriate text can be prepended from the following to the text that is to be displayed:

```
EVENT_SEVERITY_DEBUG:
EVENT_SEVERITY_INFO:
EVENT_SEVERITY_NOTICE:
EVENT_SEVERITY_WARNING:
EVENT_SEVERITY_ERROR:
EVENT_SEVERITY_ALERT:
```

Example

```
echo "EVENT_SEVERITY_ERROR:An error line" > /var/spool/cmd/eventbucket
```

The preceding example displays an output notification in the Base View event viewer as shown in figure 10.36:

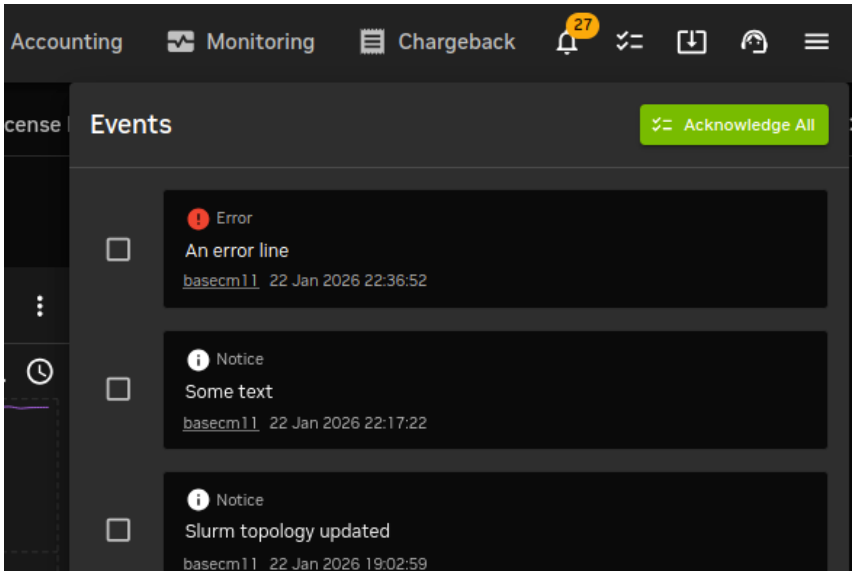


Figure 10.36: Base View Monitoring: Event Bucket Message Example

Event Bucket Filter

Regex expressions can be used to conveniently filter out the user-defined messages that are about to go into the event bucket from the shell. The filters used are placed in the event bucket filter, located by default at `/cm/local/apps/cmd/etc/eventbucket.filter`.

Event Bucket CMDaemon Directives

The name and location of the event bucket file and the event bucket filter file can be set using the `EventBucket` and `EventBucketFilter` directives from the CMDaemon configuration file directives (Appendix C).

Adding A User-Defined Message To A Device State With The Event Bucket

While the event bucket is normally used to send a message to the event viewer, it can instead be used to add a message to the state of a device. The line passed to the `echo` command then has the message and device specified in the following format:

`STATE.USERMESSAGE[.`*`device`*`]:[`*`message`*`].`

The device can be anything with a status property, such as, for example, a node, a switch, or a chassis.

Example

```
echo "STATE.USERMESSAGE.node001:just right" > /var/spool/cmd/eventbucket
```

The state then shows as:

```
cmsh -c "device ; status node001"
node001 ................. (just right) [   UP   ]
```

If the device is not specified, then the current host of the shell that is executing the `echo` command is used. For example, running these commands from the head node, `basecm11`, as follows:

Example

```
echo "STATE.USERMESSAGE:too hot" > /var/spool/cmd/eventbucket
ssh node001 'echo "STATE.USERMESSAGE:too cold" > /var/spool/cmd/eventbucket'
```

yields these states:

```
cmsh -c "device ; status basecm11"
basecm11 .................. (too hot) [   UP   ]
cmsh -c "device ; status node001"
node001 .................. (too cold) [   UP   ]
```

The added text can be cleared with echoing a blank message to that device. For example, for `node001` that could be:

```
echo "STATE.USERMESSAGE.node001:" > /var/spool/cmd/eventbucket
```

Reloading CMDaemon Logging Configuration With Event Bucket

CMDemon logging configuration is reloaded when `CMDaemon` is restarted (`systemctl restart cmd`). The logging configuration can also be reloaded without restarting CMDaemon by triggering the event bucket:

Example

```
[root@basecm11 etc]# echo LOGGING.RELOAD.CONFIG > /var/spool/cmd/eventbucket
```

Using An Event Bucket During The Node-installer Stage

The node-installer runs before systemd is up on the node that is being provisioned. This means that CMDaemon is also not yet running on that node, so that the regular event bucket features are not available during that time. However, a simplified event bucket—the node-installer event bucket—is available during this stage.

The node-installer event bucket can be particularly useful if debugging larger initialize and finalize scripts (Appendix E).

To use it, text is echoed to `/tmp/eventbucket` within the node or category scripts. The text will show up (if permitted) within the sessions of `cmsh`, and within the events viewer of Base View.

There are two different modes for the node-installer event bucket:

1. Device status `info-message` updater mode:

   Example

   ```
   echo "info-message: this text will be shown in the device status" > /tmp/eventbucket
   ```

2. Warning event mode:

   Example

   ```
   echo "Some text that will become an event" > /tmp/eventbucket
   ```

An Alternative To InfoMessages With The REST API

A cleaner alternative to InfoMessages for status messages is the `Status` REST API call (section 4.2.1 of the *Developer Manual*).

10.11 Monitoring Location With GNSS

GNSS (Global Navigation Satellite System) is the term given to GPS and similar systems. GNSS can be used to allow devices with the appropriate GNSS hardware to work out their location. The hardware is commonly implemented as a PCI-X card. A use case for this is to allow an engineer to walk to a node with a mobile phone, or to determine a sensible provisioning host.

The hardware requires the ability to receive satellite signals via an antenna. From the signals, the time of receipt and location can be worked out. BCM makes the results available in the `locations` submode of the `base` partition of `cmsh`:

Example

```
[basecm11->partition[base]]% locations
Type             Entity                 Age      Latitude    Longitude   Height   Message
---------------- ---------------------- -------- ----------- ----------- -------- ----------------
EdgeSite         Amsterdam West         2d 17h   52.1904     4.939       0        Amsterdam-South
EdgeSite         Vault-tec              6d 7h    40.5538     -105.0849   0        Fort-Collins
HeadNode         Weyland-Yutani         57m 3s   35.6586     139.7454    0        Tokyo
PhysicalNode     beta-gate-director     2d 16h   71.9022     -101.3641   0        Atlantis
PhysicalNode     omni-consumer-node001  2d 17h   41.8789     -87.6359    0        Chicago
PhysicalNode     Vault-tec-director     6d 7h    40.5538     -105.0849   0        Fort-Collins
[basecm11->partition[base]]%
```

In practice, due to environmental interference, a minimum resolution of 5m is common for longitude and lattitude. The height determination is typically 1.5x more inaccurate. Vendor specifications should be referred to for details on obtaining greater accuracy, since there are technology enhancements that can improve the accuracy.

The location is determined at start up and on demand.

10.12 Monitoring Report Queries

10.12.1 Monitoring Report Queries In `cmsh`

There are usually several hundred sources of data in BCM, and they are often of different types. An administrator would sometimes like to see the output of the data sources grouped by particular nodes. The variety of types means that examining the data output according to grouping choices would normally be awkward.

The data output can be viewed in BCM with the help of a simple query language from within the `monitoring report` submode. The query language can be used to filter by grouping choices and data values.

Data Sources For Monitoring Reports

The sources of data can be listed in `report` mode with the `fields` command:

Example

```
[basecm11->monitoring->report]% fields
Name                                Type         Values
----------------------------------- ------------ -------------------------
Accelerators                        UNDEFINED
AlertLevel                          METRIC
AverageCDULiquidDifferentialPressure METRIC
AverageCDULiquidFlow                METRIC
AverageCDULiquidReturnTemperature   METRIC
AverageCDULiquidSupplyTemperature   METRIC
AverageCDULiquidSystemPressure      METRIC
AveragePowerShelfTemperature        METRIC
AvgJobDuration                      METRIC
AvgJobStartDelay                    METRIC
BIOS Date                           SYSINFO      04/01/2014
BIOS Vendor                         SYSINFO      SeaBIOS
BIOS Version                        SYSINFO      SeaBIOS
BlockedProcesses                    METRIC
BufferMemory                        METRIC
BytesRecv                           COUNTER
BytesSent                           COUNTER
CPUGuest                            COUNTER
...
```

Filtering And Grouping For Monitoring Reports

A field filtered by a grouping can be executed using the `execute` command on a specified field, and applying a filter grouping to it using an operator.

For example, the field `ssh2node` can have the filter grouping `category` applied to it using the operator `==` for all nodes that are up. This can then be executed.

If the `ssh2node` field output is a `PASS` for nodes that are in a category `gpu`, and the `ssh2node` output is also a `PASS` for `node001` and `node002`, but a `FAIL` for `node003` and `node004`, then a report from a query executed for this, showing the non-empty groupings of hostnames could be run with:

Example

```
[basecm11->monitoring->report]% execute filter status == up group_by ssh2node category
ssh2node category Hostnames
-------- -------- ----------------------
PASS     default  node001,node002
FAIL     default  node003,node004
PASS     gpu      gpu01..gpu20
[basecm11->monitoring->report]%
```

Alternatively, the query can be saved and run as follows:

Example

```
[basecm11->monitoring->report]% add ssh2node-category
[basecm11->monitoring->report*[ssh2node-category*]% set query
```

the editor opens up and the line
`execute filter status == up group_by ssh2node category`
is entered

```
[basecm11->monitoring->report*[ssh2node-category*]% commit
[basecm11->monitoring->report[ssh2node-category]% execute
ssh2node category Hostnames
-------- -------- ----------------------
PASS     default  node001,node002
FAIL     default  node003,node004
PASS     gpu      gpu01..gpu20
```

A previously saved query can be run from outside the object with the `-q|--query` option.

```
[basecm11->monitoring->report[ssh2node-category]% exit
[basecm11->monitoring->report% execute -q ssh2node-category !#to run from outside object
```

The `help execute` text has some more detail on using the `execute` command.

Saving the list of nodes that the filter is applied to: The `--save` option takes a file base name as its argument, and saves the list of nodes that the filter applies to. Suffixes appended to the file base name are taken from the filter that is used and from the grouping values.

Example

```
[basecm11->monitoring->report]% execute filter status == up group_by ssh2node category --save /tmp/test
[basecm11->monitoring->report]% !cat /tmp/test-default-up.lst
node001
node002
```

The file name thus takes the form: *<basename>*`-`*<grouping value>*`-`*<filter value>*`.lst`
The file can be read within `cmsh` by using the operator ^

Example

```
[basecm11->monitoring->report]% device power status -n ~/tmp/PASS-default.lst
custom ..................... [   ON    ] node001
custom ..................... [   ON    ] node002
```

10.13 Monitoring With nvsm

The nvsm command in the device mode of cmsh is a BCM wrapper for the NVIDIA System Management (NVSM) software stack.

- The NVSM stack provides a CLI and API for the end user to monitor NVIDIA DGX hardware. These are documented in detail at https://docs.nvidia.com/datacenter/nvsm/latest/pdf/nvsm-user-guide.pdf.
- The nvsm wrapper command of cmsh is a front-end to some parts of the NVSM stack.

The nvsm CLI can be run directly on a node with an NVSM software stack. That CLI should not be confused with the nvsm wrapper command that is run from within the device mode of cmsh, and which is what is described in the rest of this section (section 10.13).

Running nvsm without any arguments displays the nvsm help text:

```
[basecm11->device]% nvsm
Name:
   nvsm - NVSM management

Usage:
   nvsm [OPTIONS] versions
   nvsm [OPTIONS] list

Options:
   -n, --nodes <node>
        List of nodes, e.g. node001..node015,node020..node028,node030 or ~/some/file/containing/hostnames

... many options skipped...

Examples:
   nvsm versions                       Show versions reported by NVSM for this or all nodes
   nvsm versions -c dgx-h100           Show versions reported by NVSM for the specified category of nodes
   nvsm info                           List the most recent health dumps information
   nvsm alerts                         List the alerts for this or all nodes
   nvsm health -n dgx-[001-002]        Run the NVSM dump on the specified nodes
   nvsm status -n dgx-[001-002]        Get the NVSM dump status on the specified nodes
   nvsm stop -n dgx-001                Stop the NVSM dump status on the specified nodes
```

Tab-completion prompts to nvsm suggest nvsm-specific options:

Example

```
[basecm11->device[node001]]% nvsm<TAB><TAB>
alerts    health    info      status    stop      versions
```

The nvsm-specific parts for this command are indicated by the following cmsh tree:

```
nvsm
├── alerts
│   ├── --start <start>
```

```
    └── --limit <limit>
├── health
    ├── --quick
    └── --tags <tags>
├── info
    └── --history
├── status
├── stop
└── versions
    └── --details
```

The preceding tree is discussed further next:

- `alerts`: Presents a list of the alerts that NVSM detects.

  Example

```
[basecm11->device[node001]]% nvsm alerts
Node     component_id description            event_time  message                 message_details     ...
-------- ------------ ---------------------- ----------- ---------------------- ----------------------...
node001  0            NVLink-C2C is reporti+ 1737722655  System entered degrad+ Unexpected Link Count...
node001  1            NVLink-C2C is reporti+ 1737722655  System entered degrad+ Unexpected Link Count...
node001  GPU0         GPU is reporting an e+ 1737722589  GPU0 is reporting  NV  GPU 0's NvLink link 0...
node001  GPU1         GPU is reporting an e+ 1737722589  GPU1 is reporting  NV  GPU 1's NvLink link 0...
node001  NVME0        PCI sub-system is rep+ 1737722589  System entered degrad  Device is missing on ...
node001  StorageSub+  Storage Drive configu+ 1737722595  Unsupported drive con+ Drive(s) missing or D...
```

  It has the options:

  - `--start <start>`: Sets the first index of the alert to list.

    Example

```
[basecm11->device[node001]]% nvsm alerts --start 5
```

    This skips the first 4 alerts received, and displays the rest. The alerts are not necessarily received in same order as displayed by the `nvsm alerts` command.
  - `--limit <limit>`: Sets a limit of `<limit>` alerts to be displayed.

- `health`: carries out a dump to a `.tar.xz` file for the specified node. The dump for a node named `node001` is stored in `/cm/shared/nvsm/node001` by default. It can take about 15 minutes to complete.
  - `--quick`: does a quick dump. This completes faster, but uses more memory and CPU.
  - `--tags <tags>`: a comma-separated list of tags for the dump.

- `info`: Lists the most recent health dump information.

  Example

```
[basecm11->device[node001]]% nvsm info
Node     Filename                                              Size
-------- ----------------------------------------------------- --------
node001  /cm/shared/nvsm/node001_2025-01-24-04-51-31.tar.xz  417MiB
```

The file name for a node named node001 takes a timestamped format of:

node001_YYYY-MM-DD-HH-MM-SS.tar.xz

- --history: Lists historical dumps.

  Example

```
[basecm11->device]% nvsm info --history
Node      Filename                                              Size
--------  ----------------------------------------------------  --------
node001   /cm/shared/nvsm/node001_2025-01-24-04-51-31.tar.xz   417MiB
node001   /cm/shared/nvsm/node001_2025-02-14-00-25-24.tar.xz   5.9MiB
```

- status: Shows the dump status of the node.

  Example

```
[basecm11->device[node001]]% nvsm status
Node     duration  log                                                 status   success   Result   Error
-------  --------- ----------------------------------------------------- -------- --------- ------- ------
node001  748       Jan 25 02:53:16 node001 systemd[1]: Stopping cm-nv+ active   yes       good
```

- stop: Aborts the dump creation for the specified nodes. With stop, the dump that has been created until then remains available as a directory rather than a .tar.xz file
- versions: Lists the versions of various components used by NVSM.
  - --details: May provide some more details on the component.

  Example

```
[basecm11->device[node001]]% nvsm versions
Component                           Version                         Nodes
----------------------------------- ------------------------------- -----------
FW_BMC_0                            Version Unavailable             node001
FW_CPLD_0                           Version Unavailable             node001

                                 ...many entries skipped...

cuda-driver                         12.8                            node001
datacenter-gpu-manager              1:4.0.0~10338                   node001
datacenter-gpu-manager-fabricmanager 570.59-1                       node001
dgx-release                         7.0.0                           node001
kernel                              6.8.0-31-generic-64k            node001
nvidia-driver                       570.59                          node001
nvsm                                24.09.05                        node001
os-release                          Ubuntu 24.04 LTS (Noble Numbat) node001
platform                            PG548                           node001
sbios                               02.03.13                        node001
vBIOS 0                             97.00.6c.00.03                  node001
vBIOS 1                             97.00.6c.00.03                  node001
vBIOS 2                             97.00.6c.00.03                  node001
vBIOS 3                             97.00.6c.00.03                  node001
```

The nvsm alert command can inform the cluster administrator about hardware issues more conveniently than diving into the NVSM CLI. Viewing the output of the alert may be enough to get on with solving the issue.

If that is not enough, then examining the dump file produced by nvsm health allows for further troubleshooting by experienced administrators and developers.

The dump file can be extracted with:

Example

```
root@basecm11:/cm/shared/nvsm # tar xvJf node001_2025-01-24-04-51-31.tar.xz
```

The extracted directories and files have the following layout if viewed at a two-level depth with tree -L2:

```
├── boot
│   ├── System.map-6.8.0-31-generic-64k
│   └── System.map-6.8.0-51-generic-64k
├── etc
│   ├── apt
│   ├── cm-release
│   ├── debian_version
│   ├── dgx-release
│   ├── environment
│   ├── issue
│   ├── lsb-release
│   ├── network
│   ├── nvsm
│   └── os-release
├── nvsmhealth_commands
│   ├── bash_-c_ulimit_-a_
│   ├── bash_--version
│   ├── cat_sys_devices_virtual_dmi_id_bios_version
│   ├── cat_sys_devices_virtual_dmi_id_product_name
```

... hundreds of health commands skipped ...

```
│   ├── top_-b_-n_5
│   ├── uname_-a
│   ├── uptime_-p
│   ├── _usr_bin_nv-disk-encrypt_info
│   ├── virsh_list_--all
│   ├── xl_info
│   ├── xrandr_--verbose
│   └── xset_-q
├── nvsm_resources.json
├── nvsm_show_health.json
├── proc
│   ├── cmdline
│   ├── cpuinfo
│   ├── driver
│   ├── fs
│   ├── interrupts
│   ├── iomem
│   ├── loadavg
│   ├── mdstat
│   ├── meminfo
```

```
│   ├── modules
│   └── version
├── result.json
├── usr
│   └── share
└── var
    ├── crash
    └── log
```

10.14 Monitoring With nvdebug

The NVDebug package, nvdebug, can be installed with the package manager.

The nvdebug command in the device mode of cmsh is a BCM wrapper for the NVIDIA NVDebug log collection tool. NVDebug collects diagnostic logs from NVIDIA hardware, including host logs, BMC/Redfish data, IPMI data, and health checks.

The collected logs are stored on shared storage under /cm/shared/nvdebug/<*hostname*>.

Running nvdebug without any arguments displays the nvdebug help text:

```
[basecm11->device]% nvdebug
Name:
          nvdebug - NVDebug log collection

Usage:
          nvdebug [OPTIONS] start
          nvdebug [OPTIONS] stop
          nvdebug [OPTIONS] status
          nvdebug [OPTIONS] history
          nvdebug [OPTIONS] version
          nvdebug [OPTIONS] list

Options:
          -n, --nodes <node>
               List of nodes, e.g. node001..node015,node020..node028,node030

          -t, --platform <type>
               Platform type: DGX, arm64, x86_64, NVSwitch, HGX-HMC

... many options skipped...
```

The nvdebug subcommands are:

- start: Starts NVDebug log collection on the specified nodes. The logs are collected using a systemd service and stored in /cm/shared/nvdebug/<*hostname*>.

  Example

  ```
  [basecm11->device]% nvdebug start -n node001
  Device      directory                  service                     success   Result   Error
  ----------- -------------------------- --------------------------- --------- -------- -------
  node001     /cm/shared/nvdebug/node001  cm-nvdebug@node001.service  yes       good
  ```

  The start subcommand has the options:

  - -t, --platform <type>: Platform override. Values: DGX, arm64, x86_64, NVSwitch, HGX-HMC. If not specified, the platform is derived from the node's baseboard type.

  - -j, --vendor_file <path>: Vendor JSON file for specific log collection.
  - -S, --cids <id> [id ...]: Run specific collector IDs. Mutually exclusive with -g.
  - -g, --loggroup <group>: Run collectors in a group: HealthCheck, Host, IPMI, Redfish, SSH.
  - --debug: Run nvdebug in verbose mode.

- stop: Stops nvdebug log collection on the specified nodes. The output includes the path to the (possibly incomplete) log file.

  Example

```
[basecm11->device]% nvdebug stop -n node001
Device   directory                  incomplete_log                    success    Result   Error
-------- -------------------------- --------------------------------- ---------- -------- -------
node001  /cm/shared/nvdebug/node001                                   yes        good
node001  /cm/shared/nvdebug/node001  /cm/shared/nvdebu..._12_49_22.zip  yes        good
```

- status: Shows the status of nvdebug instances. When run without -n, it lists all instances. While a collection is running, status shows additional details such as CPU usage, memory, duration, and PID.

  Example

```
[basecm11->device]% nvdebug status
Device      directory                    status     success   Result   Error
----------- ---------------------------- ---------- --------- -------- ------
basecm11    /cm/shared/nvdebug/basecm11  inactive   yes       good
node001     /cm/shared/nvdebug/node001   inactive   yes       good
...
```

  The --json option shows the raw JSON output. While active, additional fields are included:

  Example

```
[basecm11->device]% nvdebug status -n node001 --json
{
  "cpu": 0.076,
  "directory": "/cm/shared/nvdebug/node001",
  "duration": 16,
  "memory": 8261632,
  "pid": 205704,
  "status": "active",
  "success": true,
  "tasks": 1
}
```

- history: Lists collected nvdebug log files. When run without -n, it lists logs for all nodes. When a node is specified, it lists logs for that node only. For each collection run, only the archive file (.zip, .tar, and so on) is shown.

  Example

```
[basecm11->device]% nvdebug history -v
Device   directory                   files                              success    Result  Error
-------- --------------------------- ---------------------------------- ---------- ------- ------
node001  /cm/shared/nvdebug/node001  nvdebug_logs_28..._12_49_22.zip,  yes  good
                                     nvdebug_logs_28..._12_43_00.zip,
                                     nvdebug_logs_28..._10_13_50.zip,
                                     nvdebug_logs_27..._12_28_27.zip
node002  /cm/shared/nvdebug/node002  nvdebug_logs_28..._12_48_51.zip,  yes  good
                                     nvdebug_logs_28..._12_42_37.zip
node003  /cm/shared/nvdebug/node003  nvdebug_logs_28..._12_49_59.zip,  yes  good
                                     nvdebug_logs_28..._12_42_08.zip
```

The `--json` option shows the raw JSON output:

Example

```
[basecm11->device]% nvdebug history -n node001 --json
{
  "directory": "/cm/shared/nvdebug/node001",
  "files": [
    "nvdebug_logs_28_02_2026_12_49_22.zip",
    "nvdebug_logs_28_02_2026_12_43_00.zip",
    "nvdebug_logs_28_02_2026_10_13_50.zip",
    "nvdebug_logs_27_02_2026_12_28_27.zip"
  ],
  "success": true
}
```

- `version`: Shows the installed `nvdebug` version.

  Example

  ```
  [basecm11->device]% nvdebug version
  NVDebug Tool Version 1.8.0
  Build date: Fri Aug 8 01:15:02 PST 2025
  ```

- `list`: Lists available collectors for a given platform. Requires the option: `-t|--platform`.

  Example

  ```
  [basecm11->device]% nvdebug list -t arm64
  ```

  It has the options:

  - `-t, --platform <type>`: Required. Platform type: `DGX`, `arm64`, `x86_64`, `NVSwitch`, `HGX-HMC`.
  - `-g, --loggroup <group>`: Filter by log group: `HealthCheck`, `Host`, `IPMI`, `Redfish`, `SSH`.

11

Monitoring: Job Monitoring

11.1 Job Metrics Introduction

Most HPC administrators set up device-centric monitoring to keep track of cluster node resource use. This means that metrics are selected for devices, and the results can then be seen over a period of time. The results can be viewed as a graph or data table, according to the viewing option chosen. This is covered in Chapter 10.

The administrator can also select a job that is currently running, or that has recently run, and get metrics for nodes, memory, CPU, storage, and other resource use for the job. This is known as *job monitoring*, which is, as the term suggests, about job-centric rather than device-centric monitoring. Job monitoring is covered in this chapter, and uses *job metrics*.

For perspective, monitoring as discussed until now has been based on using devices or jobs as the buckets for which resource use values are gathered. Administrators can also gather, for resources consumed by jobs, the resources used by users (or any other classifier entity) as the buckets for the values, with the help of promQL-based queries. This is typically useful for watching over the resources used by a user (or other classifier entity) when jobs are run on the cluster. User-centric monitoring—or more generally, PromQL-based classifier-centric monitoring—for jobs is termed *job accounting* and is covered in Chapter 12.

11.2 Job Metrics With Cgroups

Job metrics collection uses `control groups` (`cgroups`), (section 7.10). Each job is associated with a specific cgroup that is created in each of the three base cgroups that are associated with particular cgroup controllers. The cgroup controllers are kernel components that allow metrics to be collected for processes. The PIDs of these processes are in the cgroups `tasks` file.

NVIDIA Base Command Manager 11 uses the following cgroup controllers:

- `blkio`: provides block device metrics,
- `cpuacct`: provides CPU usage metrics,
- `memory`: provides memory usage metrics.

In NVIDIA Base Command Manager before version 9.1, each job had to be put by a workload manager into a unique cgroup. However, from NVIDIA Base Command Manager 9.1 onward, this no longer necessary. By default, BCM still configures all supported workload managers to run jobs in cgroups, but it is now CMDaemon that manages the cgroup life cycle. Thus, CMDaemon ensures that:

- the necessary cgroups are created per job
- ensures that the cgroups are removed after the job is finished
- and that the last values of the metrics are collected.

Even if the administrator completely disables cgroups management in the workload manager, CMDaemon can still create and remove the three cgroups associated with the job, with each of those cgroups associated with one of the three previously-mentioned cgroup controllers.

If the workload manager creates some (or all three) cgroups for a job, then CMDaemon does not try to recreate the cgroup, but does take charge of the removal of cgroups.

In NVIDIA Base Command Manager before version 9.1, `cm-wlm-setup` configured systemd to use a joined cgroup with the following parameter settings:

Example

```
[root@node001 ~]# grep JoinControllers /etc/systemd/system.conf
JoinControllers=blkio,cpuacct,memory,freezer
[root@node001 ~]#
```

Currently this is not needed. However, if this setting remains, then CMDaemon can still collect job metrics. In order to reset the cgroup layout to the default one, the administrator can run:

```
cm-wlm-setup --reset-cgroups
```

This command removes the `JoinControllers` parameter and regenerates initrd. A reboot of the nodes is required after this.

When a job is started CMDaemon detects all the job processes. CMDaemon then ensures that the required cgroups are created, and allocates the detected processes to those cgroups. CMDaemon does not configure the cgroups in any way—this is the responsibility of the workload manager.

The tables in Appendix G.1.8 list the job metrics that BCM can monitor and visualize.

If job metrics are set up (section 11.4), then:

1. on virtual machines, block device metrics may be unavailable because of virtualization.
2. for now, the metrics are retrieved from cgroups created by the workload manager for each job. When the job is finished the cgroup is removed from the filesystem along with all the collected data. Retrieving the jobs metric data therefore means that CMDaemon must sample the cgroup metrics before the job is finished. If CMDaemon is not running during a time period for any reason, then the metrics for that time period cannot be collected, even if CMDaemon starts later.
3. block device metrics are collected for each block device by default. Thus, if there are N block devices, then there are N collected block device metrics. The monitored block devices can be excluded by configuration as indicated in section 11.4.

11.3 Job Information Retention

Each job adds a set of metric values to the monitoring data. The longer a job runs, the more data is added to the data. By default, old values are cleaned up from the database in order to limit its size. In NVIDIA Base Command Manager 11 there are several advanced configuration directives to control the job data retention, with names and default values as follows:

Advanced Configuration Directive	Default value	Unit
`JobInformationDisabled`	0	
`JobInformationKeepDuration`	2419200	s
`JobInformationKeepCount`	8192	
`JobInformationMinimalJobDuration`	0	s
`JobInformationFlushInterval`	600	s

These directives are described in detail in Appendix C, page 934.

11.4 Job Metrics Sampling Configuration

Job metrics sampling can be configured to varying degrees. For clusters where hundreds of thousands of jobs are run in a day it often makes little sense to monitor jobs, and it is often helpful to disable the `JobSampler` and `JobMetadataSampler` data producers:

Example

```
[basecm11]% monitoring setup
[basecm11->monitoring->setup]% set jobsampler disabled yes
[basecm11->monitoring->setup]% set jobmetadatasampler disabled yes
[basecm11->monitoring->setup]% commit
```

An alternative is to use the equivalent CMDaemon directive `JobInformationDisabled`, as explained on page 649.

If however CMDaemon is to keep the monitoring data, then the collection of job metrics is carried out from the cgroups in which a job runs. The administrator can tune some low level metric collection options for the `JobSampler` data producer in the `jobmetricsettings` submode:

Example

```
[basecm11]% monitoring setup
[basecm11->monitoring->setup]% use jobsampler
[basecm11->monitoring->setup[JobSampler]% jobmetricsettings
[basecm11->monitoring->setup[JobSampler]->jobmetricsettings]% show
Parameter                      Value
------------------------------ ------------------------------------------------
Revision
Enable advanced metrics        no
Map jobs to GPUs               yes
Exclude devices
Include devices
Exclude metrics
Include metrics
Exclude scripts
Include scripts
CGroup base directory          /sys/fs/cgroup
Extra scripts directory        /cm/local/apps/cmd/scripts/metrics/jobs
Keep alive sleep               8w
Sampling Type                  Both
Pickup interval                5s
Pickup times                   12
Pickup priority                50
```

The configuration parameters are:

Parameter Name	Description
`Exclude devices`	Block devices for which job metrics will not collect metrics

...continues

...continued

Parameter Name	Description
Include devices	If the list is not empty then only block device metrics for these devices will be collected, while for other devices the metrics will be skipped
Enable advanced metrics	Indicates whether advanced job metrics should be enabled (default: no)
Map jobs to GPUs	Associate the job with GPUs where the job processes run, where possible (default: yes)
Exclude metrics	List of metric names that should not be collected
Include metrics	List of metric names that should be added to metric collection
Sampling Type	Type of metric sampling (default: both). The value can be: • bright (disables Prometheus sampling by BCM) • prometheus (disables Bright sampling by BCM) • both
CGroup base directory	Cgroup base directory (default: /sys/fs/cgroup)
Keep alive sleep	Time the cgroup keepalive process sleeps (default: 8 weeks)
Pickup interval	Initially higher pickup interval (default: 5s). By default this settles down to the normal pickup interval (with a default of 120s) after the value of Pickup times has been exceeded.
Pickup times	Number of times to apply the initially higher pickup interval (default: 12)
Pickup priority	Priority of the pickup interval change (default: 50)

The amount of monitoring data gathered can also be reduced by reducing the Maximal age and Maximal samples for data producers (section 10.4.1) to smaller, but still non-zero values. A way to do this is described in section 14.8.4.

11.4.1 The Job Metrics Collection Processing Mechanism

The `cm-cgroup-job-keepalive` Process

From NVIDIA Base Command Manager 9.0 onward, when a WLM job starts, CMDaemon tracks the moment it starts and finishes, and is able to collect metrics for it. As part of this enhanced jobs metrics collection, a keepalive process, `cm-cgroup-job-keepalive`, is run for each job. Each keepalive process is a temporary process, and is added by CMDaemon to the same cgroup that the original job was placed in by the WLM.

The process `cm-cgroup-job-keepalive` itself does no work. It sleeps, and by existing it prevents its cgroup being deleted.

After a job is finished, the workload manager would normally remove the related cgroup. However the existence of the `cm-cgroup-job-keepalive` process prevents the deletion. This allows CMDaemon to collect the very last metrics data for the job from the cgroup when the job finishes. The CMDaemon then stops the `cm-cgroup-job-keepalive` process, and the cgroup is then removed because it is no longer needed.

When CMDaemon starts the `cm-cgroup-job-keepalive` process for a job, it passes the appropriate job ID, and how long it can run, in its command line options. Those values are not used by the `cm-cgroup-job-keepalive` process itself, but they are convenient for seeing what job the process is running for, and how long the job has run since it was started. For example, for a Slurm job with id 2 the running cgroup keeper process could look like:

Example

```
[root@node001 ~]# ps auxf | tail -n 10 | cut -b18-45 --complement
root      1954  0  Sl   18:12   0:03  _ /cm/local/apps/cmd/sbin/cmd -s -n -P /var/run/cmd.pid
root      2632  0  Ss   18:15   0:00      _ /cm/local/apps/cmd/sbin/cm-cgroup-job-keepalive --job 2 8w
root      2241  0  S    18:13   0:00 /cm/shared/apps/slurm/18.08.4/sbin/slurmd
root      2627  0  Sl   18:15   0:00 slurmstepd: [2.batch]
cmsuppo+  2631  0  S    18:15   0:00  _ /bin/bash /cm/local/apps/slurm/var/spool/job00002/slurm_script
cmsuppo+  2642  0  S    18:15   0:00      _ /cm/shared/apps/stresscpu/current/stresscpu2
cmsuppo+  2643  0  S    18:15   0:00          _ /cm/shared/apps/stresscpu/current/stresscpu2
cmsuppo+  2644 98  R    18:15   4:46              _ /cm/shared/apps/stresscpu/current/stresscpu2
[root@node001 ~]#
```

(The `cut` command is just used in the example to cut out the middle bits of the output so that it fits the page format well).

The `Keep Alive Sleep` Time

By default the cgroup keeper process stops after 8 weeks. This value should be increased if the jobs that are expected to run will take longer than 8 weeks. The value can be set in the `Keep Alive Sleep` parameter of the job metrics settings. If a job runs for longer than the value of `Keep Alive Sleep`, then CMDaemon cannot collect the very last metrics (from around the time that the job has finished). However all other metrics will be collected for the job as expected, even if the job running time exceeds the `Keep Alive Sleep` time.

The `OOB intervals` Parameter

When metric collection for a new job has just started, CMDaemon samples more frequently than later on. This more frequent sampling behaviour is defined by the parameter `OOB intervals` (out of band sampling interval) in the data producer configuration. In the case of job metrics collection this more frequent sampling behaviour is in `JobSampler`.

By default, the sampling interval returns to the standard `Interval` value (with a default value of 120s), as defined in the data producer settings, after the value of `Pickup times` (with a default value of 12) has been exceeded.

The parameter `Exclude Metrics` can be used to exclude metrics that are currently enabled. For example, if advanced metrics collection is enabled then `Exclude Metrics` allows either default or advanced metrics to be excluded by name.

11.5 Job Monitoring In `cmsh`

The following commands are associated with monitoring job measurables within `jobs` submode (`cmsh > wlm<[`*workload manager*`]> > jobs`, section 7.7):

The `measurables` Command

A list of job-associated measurables can be seen in the jobs submode (`cmsh > [<`*workload manager*`]> > jobs`) using the `measurables` command with a job ID. For example (much output elided):

Example

```
[basecm11->wlm[slurm]->jobs]% measurables 26
blkio.io_service_bytes_total
...
memory.usage
[basecm11->wlm[slurm]->jobs]%
```

A list of node-associated measurables can also be seen if the `-n` option is used (much output elided):

Example

```
[basecm11->wlm[slurm]->jobs]% measurables -n 26
...
gpu_power_usage:gpu0
...
memory.usage
```

The `filter` Command

The `filter` command uses options to provide filtered historic job-related information. It does not provide measurables data. The command and its options can be used to:

- retrieve running, pending, failed or finished jobs information
- select job data using regular expressions to filter by job ID
- list jobs by user name, user group, or workload manager

Running `filter` without options simply lists an unfiltered list.

Filtering on a job name can be done with the `-n|--name` option, and the `--limit` option can be used to limit the number of results displayed:

Example

```
[basecm11->wlm[slurm]->jobs]% filter -n mgbench --ended --limit 2
Job ID  Job name User  Queue Submit time     Start time      End time        Nodes   Exit code
------ -------- ----- ----- --------------- --------------- --------------- ------- ---------
26     mgbench  alice defq  May 14 11:21:41 May 14 11:36:36 May 14 11:50:24 node001 0
27     sleep    bob   defq  May 14 11:23:00 May 14 11:50:24 May 14 12:00:25 node001 0
[basecm11->wlm[slurm]->jobs]%
```

The data shown is retrieved from the running workload manager, as well as from the accounting file or database maintained by the workload manager.

Further details on the options to `filter` can be seen by running `help filter`.

The info Command

A handy command to obtain job information is info, followed by the job number:

Example

```
[basecm11->wlm[slurm]->jobs]% info 25
Parameter                      Value
------------------------------ ------------------------------------------------
Job ID                         25
Revision
Job name                       alloc-bright_cm_cluster-tools_triple-i-my-parent-i+
Job array ID
Job task ID
User                           dennis
Group                          dennis
Account                        bright_cm_cluster-tools_triple-i
Accounting info                < not set >
Workload manager cluster       slurm
Queue                          defq
Nodes                          node003
Submit time                    23/01/2026 07:41:11
Start time                     23/01/2026 07:41:11
End time                       23/01/2026 07:42:14
Run time                       1m 3s
Persistent                     no
Exit code                      0
Status                         COMPLETED
Requested CPUs                 1
Requested CPU cores            0
Requested GPU                  0
Requested memory               3.2GiB
Requested slots                0
Monitoring                     yes
Comment                        #my-parent-id __model:my-model
Run directory                  /home/dennis/war
Stdin file                     /dev/null
Stdout file                    /home/dennis/war/alloc-25.out
Stderr file                    /home/dennis/war/alloc-25.err
Total power usage              < not set >
Total GPU power usage          < not set >
Total CPU power usage          < not set >
Total power under allocation   < not set >
QOS                            normal
[basecm11->wlm[slurm]->jobs]%
```

The dumpmonitoringdata Command

The dumpmonitoringdata command displays data for measurables in the jobs submode (cmsh > [<*workload manager*]> > jobs). It is a very similar to the dumpmonitoringdata command for measurables in device mode (section 10.6.4). The main difference in the behavior of dumpmonitoringdata for these modes is that:

- In the jobs submode it shows monitoring data over a period of time for a specified job ID
- In device mode (cmsh > device) it shows monitoring data over a period of time for a specified device.

A less obvious difference is that:

- In the `jobs` submode the start and end time for the monitoring data for the job does not need to be specified. By default the start and end time of the job is assumed.
- In `device` mode the start and end time for the monitoring data for the device must be specified

The usage of the `dumpmonitoringdata` command for job measurables is:

`dumpmonitoringdata` [*OPTIONS*] [<*start-time*> <*end-time*>] <*measurable*> <*job ID*>

Options allow measurables to be retrieved and presented in various ways, including by maximum value, raw or interpolated data, and human-friendly forms. The user can also specify custom periods for the options.

For example, a historical job with job ID 4 that uses nodes `node001` and `node002` might display output for the job-associated measurable `memory.usage` as follows:

Example

```
[basecm11->wlm[slurm]->jobs]% dumpmonitoringdata memory.usage 4
# Start        - Fri Jan 23 07:30:39 2026 (1769182239)
# End          - Fri Jan 23 07:30:45 2026 (1769182345)
# Nodes        - node001,node002
# memory.usage - The total amount of memory currently being used
Entity       Timestamp                  Value      Info
------------ -------------------------- ---------- ----------
node001      2026/01/23 07:30:39.683    840 KiB
node001      2026/01/23 07:30:40.682    233 MiB
node001      2026/01/23 07:30:41.682    562 MiB
node001      2026/01/23 07:30:42.682    596 MiB
node001      2026/01/23 07:30:43.682    596 MiB
node001      2026/01/23 07:30:44.682    105 MiB
node001      2026/01/23 07:30:45.682    2.17 MiB
node002      2026/01/23 07:30:40.049    808 KiB
node002      2026/01/23 07:30:42.048    633 MiB
node002      2026/01/23 07:30:44.048    201 KiB
[basecm11->wlm[slurm]->jobs]%
```

The start and end times are optional, so specifying them is typically unnecessary. If they are not specified, then the data values that were found over the entire period of the job run are displayed.

That job with ID 26, for the example used in this section 11.5, happens to be `mgbench`, a GPU benchmarking program that runs on nodes. So displaying output for the node-associated measurable, `gpu_power_usage`, during the job run can also be useful:

Example

```
[basecm11->wlm[slurm]->jobs]% dumpmonitoringdata gpu_power_usage:gpu0 26
Start: Thu May 14 11:36:36 2020
End:   Thu May 14 11:50:24 2020
Nodes: node001
Timestamp                  Value      Info
-------------------------- ---------- ----------
2020/05/14 11:36:36        no data
2020/05/14 11:37:03.883    195.58 W
...
2020/05/14 11:50:23.887    22.224 W
2020/05/14 11:50:24        22.2241 W
[basecm11->wlm[slurm]->jobs]%
```

The data is shown per node if the job uses several nodes.

Further details of the options to `dumpmonitoringdata` for job metrics can be seen by running `help dumpmonitoringdata` within `jobs` mode.

The `statistics` Command

The `statistics` command shows basic statistics for historical job information. It allows statistics to be filtered per user or user group, and workload manager. The statistics can be grouped by hour, day, week or a custom interval.

Example

```
[basecm11->wlm[slurm]->jobs]% statistics
Queued     Running    Finished   Error      Nodes
---------- ---------- ---------- ---------- ----------
24         1          25         4          34
[basecm11->wlm[slurm]->jobs]%
```

Further details of the options to `statistics` can be seen by running `help statistics`.

12

Monitoring: Job Accounting

12.1 Introduction

In addition to the concept of metrics for devices (Chapter 10), or the concept of metrics for jobs (Chapter 11), there is also the concept of metrics gathered for a classifier entity, for resources used during jobs. This last one is typically metrics gathered per user, for resources used during jobs.

Classifier-based metrics gathering for jobs can use classification done with PromQL queries on labeled entities (sections 12.2- 12.7), or it can be done with CMDaemon database queries (section 12.8).

Classifier-based metrics gathering for jobs is more conveniently called *job accounting*, partly because it ressembles the idea of an accountant watching over users to track their resource use while they carry out their jobs.

The concept, implementation, analysis, and visualization of job accounting are described in this Chapter.

For example, in BCM jobs resource usage can be presented per user. Thus, if there are jobs in a queue that are being processed, then the jobs can be listed:

Example

```
[basecm11->wlm[slurm]->jobs]% list | head
Type         Job ID       User   Queue   Running time Status     Nodes
------------ ------------ ------ ------- ------------ ---------- ------------------
Slurm        1325         tim    defq    1m 2s        COMPLETED  node001..node003
Slurm        1326         tim    defq    1m 1s        COMPLETED  node001..node003
Slurm        1327         tim    defq    1m 2s        COMPLETED  node001..node003
Slurm        1328         tim    defq    32s          RUNNING    node001..node003
Slurm        1329         tim    defq    0s           PENDING
```

The resource usage statistics gathered per user, for example for a user `tim`, can then be analyzed and visualized using the job accounting interface of Base View (section 12.5).

12.2 Labeled Entities

In job accounting, job metrics during a run are tagged with extra labels, such as the job ID, host name, and the user running the job. The modified job metrics object that is tagged in this way then becomes a job accounting-related object, called a labeled entity.

Administrators interested in using job accounting can simply skip ahead and start reading about the Base View job accounting interface in 12.5, and just explore it directly. Those who would prefer some background on how job accounting is integrated with BCM and PromQL, can continue reading this section (12.2) and the next one (12.3).

12.2.1 Dataproducers For Labeled Entities

To view labeled entities in cmsh, the path to the labeledentity submode is:

cmsh > monitoring > labeledentity

The labeledentity submode allows job accounting-related objects, called labeled entities, to be viewed. The labels are in the form <*key*>="<*value*>", for example: hostname="node001", or user="alice".

The default, existing labeled entities are created from the built-in JobSampler and JobMetadataSampler dataproducers when a job is run. Custom samplers, of type prometheus, can be used to create further custom labeled entities. A custom sampler dataproducer, for example customsamplerextras, can be created from the monitoring setup mode of cmsh as follows:

Example

```
[basecm11]% monitoring setup
[basecm11->monitoring->setup]% add prometheus customsamplerextras
[basecm11->monitoring->setup*[customsamplerextras*]]%
```

The customsamplerextras dataproducer can now have its properties configured and committed as described in section 10.5.4.

12.2.2 PromQL And Labeled Entities

Labeled entities can be used by administrators to help create and debug job-related queries in the Prometheus query language, PromQL. PromQL is a part of the Prometheus monitoring and alerting toolkit (https://prometheus.io). Basic PromQL documentation is available at https://prometheus.io/docs/prometheus/latest/querying/basics/.

12.2.3 Job IDs And Labeled Entities

Each job ID has a number of labeled entities associated with it. Since the number of labeled entities scales with the number of nodes and jobs, the number of labeled entities can be very large. Therefore, if examining these entities using the CMDaemon front ends such as cmsh or Base View, then filtering or sorting the output is useful. For example, labeled entities associated with node001, and with the JobSampler data producer, and with job 1329 from the preceding output, could be viewed by filtering the full list of labeled entities as follows (output truncated and ellipsized):

Example

```
[basecm11->monitoring->labeledentity]% list|head -2; list|grep 'job_id="1329"' |grep node001 |grep JobSampler
Index  Name (key)                                                                              ...
------ ----------------------------------------------------------------------------------------...
45446  hostname="node001",job="JobSampler",job_id="1329",wlm="slurm"                           ...
45447  device="vda",hostname="node001",job="JobSampler",job_id="1329",wlm="slurm"              ...
45448  device="vda",hostname="node001",job="JobSampler",job_id="1329",mode="read",wlm="slurm"...
...
```

12.2.4 Measurables And Labeled Entities

The measurables (metrics) for an entity can be listed with the measurables (or metrics) command. For a particular entity with a JobSampler property and index value of 45447, the command can be run as follows:

```
[basecm11->monitoring->labeledentity]% measurables 45447
Type    Name                               Parameter      Class       Producer
------- ---------------------------------- -------------- ----------- --------------
Metric  job_blkio_sectors                                 Prometheus  JobSampler
Metric  job_blkio_time_seconds                            Prometheus  JobSampler
[basecm11->monitoring->labeledentity]%
```

In the labeledentity mode of cmsh, the measurables listing command, which lists the measurables for labeled entities, should not be confused with the measurable navigation command, which brings the administrator to the measurable submode under the main monitoring mode.

12.3 PromQL Queries

12.3.1 The Default PromQL Queries...

By default there are several predefined PromQL queries already available. The queries can be listed from the query submode:

Example

```
[basecm11->monitoring->query]% list
Name (key)                                                  Start time End time Interval Class
----------------------------------------------------------- ---------- -------- -------- ------------------
account_job_effective_cpu_seconds                           now                 0s       accounting
account_job_io_bytes                                        now                 0s       accounting
account_job_memory_usage_bytes                              now                 0s       accounting
account_job_running_count                                   now                 0s       accounting
account_job_waiting_seconds                                 now                 0s       accounting
account_job_wall_clock_seconds                              now                 0s       accounting
account_job_wasted_cpu_seconds                              now                 0s       accounting
accounts_usage_gpu                                          now                 0s       accounting
accounts_used_gpu                                           now                 0s       accounting
accounts_wasted_memory                                      now                 0s       accounting
cluster_cpu_usage_percent                                   now-1d     now      15m      cluster
container_memory_usage_bytes                                now-1d     now      1h       container
container_network_received_bytes                            now-1d     now      1h       container
container_total_cpu_usage_secs                              now-1d     now      1h       container
container_total_fs_usage_bytes                              now-1d     now      1h       container
cpu_usage_by_cluster                                        now-1d     now      1h       kubernetes
cpu_usage_by_deployment                                     now-1d     now      1h       kubernetes
cpu_usage_by_namespace                                      now-1d     now      1h       kubernetes
fs_usage_by_cluster                                         now-1d     now      1h       kubernetes
fs_usage_by_deployment                                      now-1d     now      1h       kubernetes
fs_usage_by_namespace                                       now-1d     now      1h       kubernetes
gpu_power_usage_for_account                                 now                 0s
gpu_power_usage_for_job_id                                  now                 0s
gpu_power_usage_for_job_name                                now                 0s
gpu_power_usage_for_user                                    now                 0s
groups_job_allocated_nodes                                  now-1d     now      15m      jobs
groups_job_cpu_usage                                        now-1d     now      15m      jobs
groups_job_io_bytes_per_second                              now-1d     now      15m      jobs
groups_job_memory_bytes                                     now-1d     now      15m      jobs
groups_job_waiting                                          now-1d     now      15m      jobs
groups_usage_gpu                                            now                 0s       accounting
groups_used_gpu                                             now                 0s       accounting
job_effective_cpu_seconds_job_name_for_user                 now                 0s       accounting/level/1
job_gpu_workload_power_profile_for_job_id                   now                 0s       jobs
job_gpu_workload_power_profile_for_job_name                 now                 0s       jobs
job_gpu_workload_power_profile_for_one_job_id               now                 0s       jobs
job_gpu_workload_power_profile_for_user                     now                 0s       jobs
job_information_by_account                                  now                 0s       drilldown/level/0
job_information_by_job_id_for_account_and_user_and_job_name now                 0s       drilldown/level/3
job_information_by_job_id_for_user                          now                 0s       drilldown/level/1
job_information_by_job_id_for_user_and_job_name             now                 0s       drilldown/level/2
```

job_information_by_job_name_for_account	now		0s	drilldown/level/1
job_information_by_job_name_for_account_and_user	now		0s	drilldown/level/2
job_information_by_job_name_for_user	now		0s	drilldown/level/1
job_information_by_user	now		0s	drilldown/level/0
job_information_by_user_for_account	now		0s	drilldown/level/1
job_information_by_user_for_account_and_job_name	now		0s	drilldown/level/2
job_io_bytes_per_job_name_for_user	now		0s	accounting/level/1
job_memory_usage_bytes_per_job_name_for_user	now		0s	accounting/level/1
job_names_job_allocated_nodes	now-1d	now	15m	jobs
job_names_job_cpu_usage	now-1d	now	15m	jobs
job_names_job_io_bytes_per_second	now-1d	now	15m	jobs
job_names_job_memory_bytes	now-1d	now	15m	jobs
job_names_job_waiting	now-1d	now	15m	jobs
job_names_usage_gpu	now		0s	accounting
job_names_used_gpu	now		0s	accounting
job_running_count_job_name_for_user	now		0s	accounting/level/1
job_waiting_seconds_job_name_for_user	now		0s	accounting/level/1
job_wall_clock_seconds_job_name_for_user	now		0s	accounting/level/1
job_wasted_cpu_seconds_job_name_for_user	now		0s	accounting/level/1
jobs_wasted_allocated_gpus	now		0s	accounting
memory_usage_by_cluster	now-1d	now	1h	kubernetes
memory_usage_by_deployment	now-1d	now	1h	kubernetes
memory_usage_by_namespace	now-1d	now	1h	kubernetes
net_usage_by_cluster	now-1d	now	1h	kubernetes
net_usage_by_deployment	now-1d	now	1h	kubernetes
net_usage_by_namespace	now-1d	now	1h	kubernetes
power_usage_for_account	now		0s	
power_usage_for_job_id	now		0s	
power_usage_for_job_name	now		0s	
power_usage_for_user	now		0s	
unused_gpu_job_name_for_user	now		0s	accounting/level/1
used_gpu_job_name_for_user	now		0s	accounting/level/1
users_job_allocated_nodes	now-1d	now	15m	jobs
users_job_cpu_usage	now-1d	now	15m	jobs
users_job_effective_cpu_seconds	now		0s	accounting
users_job_io_bytes	now		0s	accounting
users_job_io_bytes_per_second	now-1d	now	15m	jobs
users_job_memory_bytes	now-1d	now	15m	jobs
users_job_memory_usage_bytes	now		0s	accounting
users_job_running_count	now		0s	accounting
users_job_waiting	now-1d	now	15m	jobs
users_job_waiting_seconds	now		0s	accounting
users_job_wall_clock_seconds	now		0s	accounting
users_job_wasted_cpu_seconds	now		0s	accounting
users_unused_gpu	now		0s	accounting
users_usage_gpu	now		0s	accounting
users_used_gpu	now		0s	accounting
users_wasted_allocated_gpus	now		0s	accounting
users_wasted_memory	now		0s	accounting
wasted_allocated_gpus_for_user	now		0s	accounting/level/1
wasted_memory_job_name_for_account	now		0s	accounting/level/1
wasted_memory_job_name_for_user	now		0s	accounting/level/1

The queries can be conceptually divided into their classes, which at the time of writing of this section (January 2026) are: `accounting`, `cluster`, `container`, `jobs`, `kubernetes`, along with various `drilldown` levels which classifies the queries according to various groups. The grouping for `drilldown` levels can

be confusing, and the `drilldownoverview` command (section 12.7.1) can be helpful in clarifying the query intention for these.

By default, queries of all classes are sampled over a period, except for the `accounting` and `drilldown` metrics.

A metric in the `accounting` class query is evaluated (interpolated) from existing values. These existing values are raw samples gathered over the period, up to the time when the query is evaluated.

12.3.2 ...And A Short Description Of Them

The description of each query can be listed with a little `cmsh` and unix text utility juggling:

Example

```
[basecm11->monitoring->query]% foreach * (get name; get description) | paste - - | expand -t 60
```

This yields the following table:

Table 12.3: PromQL Query Descriptions

Name	Description
`account_job_effective_cpu_seconds`	CPU seconds effectively used by account for the last period
`account_job_io_bytes`	Total I/O by account during the last period in Bytes
`account_job_memory_usage_bytes`	Total memory usage by account during the last period in Byte seconds
`account_job_running_count`	Number of jobs running by account during the last period
`account_job_waiting_seconds`	Total waiting time for account jobs in seconds during the last period
`account_job_wall_clock_seconds`	Wall clock time used by account for the last period
`account_job_wasted_cpu_seconds`	CPU seconds allocated but not used by account for the last period
`accounts_usage_gpu`	Total used GPU time grouped by account for the specified period
`accounts_used_gpu`	Used GPUs, for values of use greater than or equal to 0.1%, averaged and grouped by account using them in the specified period
`accounts_wasted_memory`	The sum of the minimal wasted memory over all nodes per account for the last period
`cluster_cpu_usage_percent`	CPU usage percentage over all nodes up
`container_memory_usage_bytes`	Containers' memory usage in bytes
`container_network_received_bytes`	Containers' total Received bytes

...continues

Table 12.3: PromQL Query Descriptions...continued

Name	Description
container_total_cpu_usage_secs	Containers' total CPU Usage in seconds
container_total_fs_usage_bytes	Containers' total filesystem usage in bytes
cpu_usage_by_cluster	CPU usage by cluster in nr. of cores
cpu_usage_by_deployment	CPU usage by deployment in nr. of cores
cpu_usage_by_namespace	CPU usage by namespace in nr. of cores
fs_usage_by_cluster	Current FS I/O by cluster in Bytes
fs_usage_by_deployment	Current FS I/O by deployment in Bytes
fs_usage_by_namespace	Current FS I/O by namespace in Bytes
gpu_power_usage_for_account	The total power usage for all jobs of a account for the last period
gpu_power_usage_for_job_id	The total power usage for a job for the last period
gpu_power_usage_for_job_name	The total power usage for all jobs of a job_name for the last period
gpu_power_usage_for_user	The total power usage for all jobs of a user for the last period
groups_job_allocated_nodes	Number of nodes allocated by groups
groups_job_cpu_usage	Effective CPU usage by groups
groups_job_io_bytes_per_second	Current I/O for group jobs in B/s
groups_job_memory_bytes	Current memory consumption for group jobs in Bytes
groups_job_waiting	Number of jobs currently waiting for every group
groups_usage_gpu	Total used GPU time grouped by group for the specified period
groups_used_gpu	Used GPUs, for values of use greater than or equal to 0.1%, averaged and grouped by groups using them in the specified period
job_effective_cpu_seconds_job_name_for_user	CPU seconds effectively used by by job_name for a user for the last period
job_gpu_workload_power_profile_for_job_id	The number of time steps a GPU workload power profile was active by a job ID
job_gpu_workload_power_profile_for_job_name	The number of time steps a GPU workload power profile was active for a job name
job_gpu_workload_power_profile_for_one_job_id	The number of time steps a GPU workload power profile was active for one job ID
job_gpu_workload_power_profile_for_user	The number of time steps a GPU workload power profile was active for a user while their job ran

...continues

Table 12.3: PromQL Query Descriptions...continued

Name	Description
job_information_by_account	Generic job information drill down query grouped by account
job_information_by_job_id_for_account_and_user_and_job_name	Generic job information drill down query grouped by wlm and job_id for a specific account, user and job_name
job_information_by_job_id_for_user	Generic job information drill down query grouped by wlm and job_id for a specific user
job_information_by_job_id_for_user_and_job_name	Generic job information drill down query grouped by wlm and job_id for a specific user and job_name
job_information_by_job_name_for_account	Generic job information drill down query grouped by job_name for a specific account
job_information_by_job_name_for_account_and_user	Generic job information drill down query grouped by job_name for a specific account and user
job_information_by_job_name_for_user	Generic job information drill down query grouped by job_name for a specific user
job_information_by_user	Generic job information drill down query grouped by user
job_information_by_user_for_account	Generic job information drill down query grouped by user for a specific account
job_information_by_user_for_account_and_job_name	Generic job information drill down query grouped by user for a specific account and job_name
job_io_bytes_per_job_name_for_user	Total I/O by job_name for a user during the last period in Bytes
job_memory_usage_bytes_per_job_name_for_user	Total memory usage by job_name for a user during the last period in Byte seconds
job_names_job_allocated_nodes	Number of nodes allocated by job name
job_names_job_cpu_usage	Effective CPU usage by job name
job_names_job_io_bytes_per_second	Current I/O for jobs in B/s
job_names_job_memory_bytes	Current memory consumption for jobs in Bytes
job_names_job_waiting	Number of jobs currently waiting for every job_name
job_names_usage_gpu	Total used GPU time grouped by job name for the specified period
job_names_used_gpu	Used GPUs, for values of use greater than or equal to 0.1%, averaged and grouped by job name using them in the specified period
job_running_count_job_name_for_user	Number of jobs running by job_name for a user during the last period

...continues

Table 12.3: PromQL Query Descriptions...continued

Name	Description
job_waiting_seconds_job_name_for_user	Total waiting time for jobs by job_name for a user in seconds during the last period
job_wall_clock_seconds_job_name_for_user	Wall clock time used by job_name for a user for the last period
job_wasted_cpu_seconds_job_name_for_user	CPU seconds allocated but not used by job_name for a user for the last period
jobs_wasted_allocated_gpus	Average % of allocated GPUs wasted for jobs that ran in the specified period, averaged and grouped by job_id
memory_usage_by_cluster	Total memory usage by cluster during the last week in Bytes per second
memory_usage_by_deployment	Total memory usage by deployment during the last week in Bytes per second
memory_usage_by_namespace	Total memory usage by namespace during the last week in Bytes per second
net_usage_by_cluster	Network usage by cluster in Bytes per second
net_usage_by_deployment	Network usage by deployment in Bytes per second
net_usage_by_namespace	Network usage by namespace in Bytes per second
power_usage_for_account	The total power usage for all jobs of a account for the last period
power_usage_for_job_id	The total power usage for all jobs of a job_id for the last period
power_usage_for_job_name	The total power usage for all jobs of a job_name for the last period
power_usage_for_user	The total power usage for all jobs of a user for the last period
unused_gpu_job_name_for_user	Unused GPUs, for values of use less than 0.1%, averaged and grouped by job names using them in the specified period, for a particular user
used_gpu_job_name_for_user	Used GPUs, for values of use greater than or equal to 0.1%, averaged and grouped by job names that ran on them in the specified period, for a particular user
users_job_allocated_nodes	Number of nodes allocated by users
users_job_cpu_usage	Effective CPU usage by users
users_job_effective_cpu_seconds	CPU seconds effectively used by users for the last period
users_job_io_bytes	Total I/O by users during the last period in Bytes
users_job_io_bytes_per_second	Current I/O for user jobs in B/s

...continues

Table 12.3: PromQL Query Descriptions...continued

Name	Description
users_job_memory_bytes	Current memory consumption for user jobs in Bytes
users_job_memory_usage_bytes	Total memory usage by users during the last period in Byte seconds
users_job_running_count	Number of jobs running by users during the last period
users_job_waiting	Number of jobs currently waiting for every user
users_job_waiting_seconds	Total waiting time for users jobs in seconds during the last period
users_job_wall_clock_seconds	Wall clock time used by users for the last period
users_job_wasted_cpu_seconds	CPU seconds allocated but not used by users for the last period
users_unused_gpu	Unused GPUs, for values of use less than 0.1%, averaged and grouped by users using them in the specified period
users_usage_gpu	Total used GPU time grouped by user for the specified period
users_used_gpu	Used GPUs, for values of use greater than or equal to 0.1%, averaged and grouped by users using them in the specified period
users_wasted_allocated_gpus	Average % of allocated GPUs wasted for jobs that ran in the specified period, averaged and grouped by user
users_wasted_memory	The sum of the minimal wasted memory over all nodes per user for the last period
wasted_allocated_gpus_for_user	Average % of allocated GPUs wasted for jobs that ran in the specified period, averaged and grouped by job_id, for a particular user
wasted_memory_job_name_for_account	The sum of the minimal wasted memory over all nodes by job_name for a account for the last period
wasted_memory_job_name_for_user	The sum of the minimal wasted memory over all nodes by job_name for a user for the last period

The listings give an idea of what the query does.

For example, for the users_job_cpu_usage utility, the idea is that it shows the CPU usage for jobs for each user.

12.3.3 Modifying The Default PromQL Query Properties

The properties of a particular query can be shown and modified:

Example

```
[basecm11->monitoring->query]% use users_job_cpu_usage
[basecm11->monitoring->query[users_job_cpu_usage]]% show
Parameter                      Value
------------------------------ ------------------------------------------------
Name                           users_job_cpu_usage
Revision
Class                          jobs
Alias
Start time                     now-1d
```

```
End time                       now
Interval                       15m
Description                    Effective CPU usage by users
PromQL Query                   <136B>
Access                         Public
Unit                           CPU
Price                          0.000000
Currency                       $
Preference                     0
Drill down                     <0 in submode>
Notes                          <0B>
```

The PromQL query code itself is typically a few lines long, and can also be viewed and modified using `get`, and `set`.

12.3.4 An Example PromQL Query, Properties, And Disassembly

The `users_job_cpu_usage` query is a standard predefined query, and is used as an example here. The query shows the CPU usage by a user around the time the sample was taken. It is sometimes called an "instantaneous" value. However it is not that instantaneous, because its value is calculated by taking samples of the CPU usage over the last 10 minutes of the job run rather than at the query time. The code for the query can be viewed with:

Example

```
[basecm11->monitoring->query]% get users_job_cpu_usage promqlquery
sum by(user) (
  irate(job_cpuacct_usage_seconds[10m])
    * on(wlm, job_id, hostname) group_right()
  (job_metadata_is_running)
)
```

For those unfamiliar with PromQL, some disassembly of the `users_job_cpu_usage` query is helpful.

Terminology used by PromQL and BCM, for the pieces used to build the query, is listed in the following table:

PromQL Terminology	Example	BCM Terminology
Query	`users_job_cpu_usage`	PromQL query
Instant query	`job_cpuacct_usage_seconds`	Metric (from JobSampler dataproducer, belonging to the Prometheus class)
Range vector	`job_cpuacct_usage_seconds[10m]`	Metric samples over a time span

As was mentioned before: job account metrics, unlike traditional metrics, are not directly associated with the device-related objects. For such metrics, the monitoring data command `dumpmonitoringdata` is therefore not accessed in `cmsh` from `device` mode or `category` mode.

Instead, the Prometheus metric `job_cpuacct_usage_seconds`, for example, is accessed via the `labeledentities` mode.

The properties and interpolated values at a particular instant of time for the metric can be accessed via an `instantquery` such as (some output excised for clarity):

Example

```
[basecm11->monitoring->labeledentity]%  instantquery job_cpuacct_usage_seconds
Name                        hostname job         job_id  user  wlm   Timestamp Value
```

```
------------------------- -------- ----------- ------- ----- ----- --------- -----
job_cpuacct_usage_seconds  node001  JobSampler  623     tony  slurm 15:52:15  129
job_cpuacct_usage_seconds  node002  JobSampler  624     tony  slurm 15:52:15  71
...
```

The properties and interpolated values of the metric over a range of time can be accessed via a `rangequery` such as (some output excised for clarity):

Example

```
[basecm11->monitoring->labeledentity]% rangequery --start now-2h --end now job_cpuacct_usage_seconds
Name                       hostname job         job_id  user  wlm   Timestamp Value
------------------------- -------- ----------- ------- ----- ----- --------- -----
job_cpuacct_usage_seconds  node001  JobSampler  623     tony  slurm 13:52:15  82
job_cpuacct_usage_seconds  node001  JobSampler  623     tony  slurm 14:52:15  97
job_cpuacct_usage_seconds  node001  JobSampler  623     tony  slurm 15:52:15  129
job_cpuacct_usage_seconds  node002  JobSampler  624     tony  slurm 13:52:15  46
job_cpuacct_usage_seconds  node002  JobSampler  624     tony  slurm 14:52:15  23
...
```

The association of the `instantquery` and `rangequery` output with job accounts is because its dataproducer is `JobSampler`.

Further options for the `instantquery` and `rangequery` commands can be found in their `help` texts within `cmsh`.

Time specifications for the `instantquery` and `rangequery` commands is covered further in section 12.3.7.

12.3.5 Aside: Getting Raw Values For A Prometheus Class Metric

The PromQL language is aimed at providing an overall view of jobs and resource usage. The actual individual raw values that Prometheus metrics are built on—the entries in the Time Series Database (TSDB)—are not regarded as being important for the end user. The emphasis in PromQL is on seeing the values as seen by statistical reworking.

This section, which is about the raw TSDB values, is thus provided as background information for administrators who would anyway like to see what the raw values look like.

Raw values of the metric for a job ID can be accessed by using the index of the labeled identity that is associated with that job ID. For example, job ID 624 can have its index found with some grepping (some output elided or excised for clarity):

Example

```
[basecm11->monitoring->labeledentity]% list|head -2; list|grep 'hostname="node' | grep 'job_id="4"'

Index    Name (key)                                Introduction  Last used
-------- ----------------------------------------- ------------ --------------
2d0b     ...,hostname="node001",job_id="4",...     09:10:24 PST  09:12:39 PST
da2b     ...,hostname="node002",job_id="4",...     09:10:24 PST  09:12:39 PST
[basecm11->monitoring->labeledentity]%
```

The index for the job ID 4 is da2b for `node002`. The job ID can be used by the `dumpmonitoringdata` command to show the series raw values along with their time stamps:

Example

```
[basecm11->monitoring->labeledentity]% dumpmonitoringdata -24h now job_cpuacct_usage_seconds da2b
# Start                      - Wed Feb 11 11:09:01 2026 (1770836941)
# End                        - Thu Feb 12 11:09:01 2026 (1770923341)
```

```
# job_cpuacct_usage_seconds - Total CPU time consumed by processes
Timestamp                  Value      Info
-------------------------- ---------- ----------
2026/02/12 09:10:24.865    0.0235s
2026/02/12 09:10:26.863    0.405s
2026/02/12 09:10:28.863    0.551s
2026/02/12 09:10:31.863    0.986s
2026/02/12 09:10:32.863    1.16s
2026/02/12 09:10:34.863    1.46s
2026/02/12 09:10:44.863    2.87s
2026/02/12 09:10:54.863    4.3s
2026/02/12 09:11:04.863    5.28s
2026/02/12 09:11:14.863    5.71s
2026/02/12 09:11:24.864    6.12s
2026/02/12 09:11:34.863    7.1s
2026/02/12 09:11:44.863    8.5s
2026/02/12 09:11:54.864    9.91s
2026/02/12 09:12:04.864    11.2s
2026/02/12 09:12:14.863    12s
2026/02/12 09:12:24.863    12.3s
2026/02/12 09:12:38.605    13.3s
2026/02/12 09:12:39.605    no data
[basecm11->monitoring->labeledentity]%
```

These raw values are the values that are used for interpolation during PromQL queries.

The label names for job samples can be seen using the index:

```
[basecm11->monitoring->labeledentity]% show da2b
Parameter      Value
-------------- ------------------------------------------------------------------------------------------
Index          da2b
Name           account="bright_cm_cluster-tools_...,hostname="node002",job_id="4"...,job_name="allo...
Revision
Timestamp      0000001770916224
Introduction   Thu, 12 Feb 2026 09:10:24 PST
Last used      Thu, 12 Feb 2026 09:12:39 PST
Permanent      no
```

12.3.6 ...An Example PromQL Query, Properties, And Disassembly (Continued)

Getting back from the aside about raw values, and continuing on with the example PromQL query from the start of this section (page 694), the query code for the query `users_job_cpu_usage` was:

```
sum by(user) (
  irate(job_cpuacct_usage_seconds[10m])
    * on(wlm, job_id, hostname) group_right()
  (job_metadata_is_running)
)
```

With the necessary background explanations having been carried out, the disassembly of this query can now be done:

The core of the query is built around the job sampler metric `job_cpuacct_usage_seconds`.

The `irate` measurement in this case calculates the rate of change based on the last two most recent values in the Prometheus range vector.

The Prometheus range vector is formed from the Prometheus instant query by using the square brackets with a time value enclosed (`job_cpuacct_usage_seconds[10m]`). The Prometheus instant query is, as the terminology table earlier pointed out, the Prometheus version of the job sampler metric.

Getting back to the range vector, a range vector in general is a series of values formed from the corresponding Prometheus instant query. With the instant query being `job_cpuacct_usage_seconds` here, the range vector is formed over a span of 10 minutes.

After the `irate` function has taken the average, the resultant is what in PromQL is called an instant vector value. This consists of data in the form {CPU seconds consumed during period, timestamp associated with time period sample}. The instant vector value is then joined against each vector element of the pair `job_id`, `hostname` to generate the labeled identifier for the job running on the node. The `group_right` of the result uses the `wlm`, `job_id` and `hostname` as the leading labels in the label identifiers. The `job_metadata_is_running` function means that values are generated only while `job_metadata` is running. The `sum by(user)` function means that the metric is aggregated over all raw data and grouped by user.

Visualisation based on the result is most easily carried out by plotting job CPU usage for each user against time in the period specified, which can be done in a more user-friendly way with Base View.

The `users_job_wall_clock_seconds` query, is similar, and can be used to plot wall clock seconds consumed by a user over the last period:

Example

```
[basecm11->monitoring->query]% get users_job_wall_clock_seconds promqlquery
sum by(user) (
  max_over_time(job_metadata_running_seconds[$period])
    * on(wlm, job_id) group_right()
  max_over_time(job_metadata_num_cpus[$period])
)
[basecm11->monitoring->query]%
```

Predefined queries can be executed in the `labeledentity` mode with the `-q` option:

```
[basecm11->monitoring->labeledentity]% instantquery -q users_job_wall_clock_seconds
# using default parameter: period=1w
Name                          user     Timestamp                 Value               Unit
----------------------------- -------- ------------------------- ------------------- --------
users_job_wall_clock_seconds  alice    Fri Feb 10 16:05:01 2023  17896.886001110077  s
users_job_wall_clock_seconds  bob      Fri Feb 10 16:05:01 2023  20670.46400117874   s
users_job_wall_clock_seconds  charlie  Fri Feb 10 16:05:01 2023  10947.136002540588  s
```

Tab-completion for the `-q` option reveals the possible Prometheus queries. Regular measurables can be viewed without the `-q` option.

If no period is specified with the `-p|--parameter` option, then the default period is `1w` as the output indicates.

If Base View is used instead of `cmsh`, then Prometheus queries can be selected via the navigation path:

Menu bar > *Accounting and reporting icon* > `Monitoring` > `PromQL Queries` > `Show/Hide query selection`

For example the `users_job_wall_clock_seconds` query can be selected, its query parameter can be set to 1 week, the change saved, and the query run.

12.3.7 Time Specification With PromQL

PromQL allows a variety of time specifications.

In Base View there is a monitoring interface dashboard display that is independent of the `cmsh` display (section 12.5), and times can be specified for range and instant queries there.

In `cmsh`, time specifications that can be set for the `rangequery` and `instantquery` queries can use some additional time specifications, as shown in the following examples:

Example

Time specification	What time is meant
now	at the time it is run
now-30m	30 minutes ago
now-1h	one hour ago
now-1h/h	an hour ago, starting at the start of that hour
now-2d	48 hours ago
now-0d/d	today's midnight (the most recently-passed midnight)*
now-1d/d	yesterday's midnight**
now-2d/d	day before yesterday's midnight
Thursday, July 11, 2019 17:00:00	an absolute time (as set by the calendar widget)

*The meaning can be understood by looking at how `/d` operates. It rounds off the day by truncation, which in the case of `now-0d/d` means the midnight prior to now.

**Similarly, yesterday's midnight is the midnight immediately prior to 24 hours ago.

The time units for PromQL are (`https://prometheus.io/docs/prometheus/latest/querying/basics/#range-vector-selectors`):

- `s` - seconds
- `m` - minutes
- `h` - hour
- `d` - day
- `w` - week
- `M` - month (31 days). `M` is not a PromQL unit, so it cannot be used inside a query. But it is a handy alias in BCM for an invariant time of 31 days.
- `y` - year

The `-` operator in Prometheus is an offset operator, used only after `now` in the time specification field. The `/`*<time unit>* syntax implies a start at that unit of time.

Thus if, for example, `now-1d/d` is set as the end time, then when the query runs, it picks up the values at "yesterday's midnight." For each user, the value is the number of seconds the job of the user was in a wait state for the week prior to yesterday's midnight. It rounds off the day by truncation, which in the case of `now-1d/d` means the midnight prior to 24 hours ago. Varying the user means that the number of seconds varies accordingly.

12.4 Parameterized PromQL Queries

It is also possible to create parameterized queries using the *<key>*="*<value>*" labels in the labeled entities mode.

This is handy for running the same query with different parameters or other drilldown options.

For example, an existing unparameterized query

```
users_job_wall_clock_seconds
```

can be used as a starting point:

```
[basecm11->monitoring->query]% get users_job_wall_clock_seconds promqlquery
sum by(user) (
  max_over_time(job_metadata_running_seconds[$period])
    * on(wlm, job_id) group_right()
  max_over_time(job_metadata_num_cpus[$period])
)
```

For convenience, the original query can be cloned over to a new, soon-to-be-parameterized, query called

`users_job_wall_clock_per_account_seconds`

using

```
[basecm11->monitoring->query]% clone users_job_wall_clock_seconds users_job_wall_clock_per_account_seconds
```

The idea is that Slurm accounts become a parameter in the new query.

Parameter fields can now be added to the query. All fields in `users_job_wall_clock_per_account_seconds` can be replaced verbatim. So any part of the query can be made into a parameter.

```
[basecm11->monitoring->query]% get users_job_wall_clock_per_account_seconds promqlquery
sum by(user) (
  max_over_time(job_metadata_running_seconds{account="${account}"}[$period])
    * on(wlm, job_id) group_right()
  max_over_time(job_metadata_num_cpus{account="${account}"}[$period])
)
```

Some further adjustments are:

```
[basecm11->monitoring->query]% use users_job_wall_clock_per_account_seconds
[basecm11->monitoring->query*[users_job_wall_clock_per_account_seconds*]]% set class accounting/level/1
[basecm11->..._seconds*]% set description "Wall clock time used by users per account for the last period"
[basecm11->monitoring->query*[users_job_wall_clock_per_account_seconds*]]% commit
```

The new query can then be run by the administrator on demand.

The original query sums over all accounts:

```
[basecm11->monitoring->labeledentity]% instantquery -q users_job_wall_clock_seconds
user      Timestamp                 Value
--------- ------------------------- -------------------
alice     Mon Jun 24 10:25:20 2019  28.787
bob       Mon Jun 24 10:25:20 2019  26.787
charline  Mon Jun 24 10:25:20 2019  102.83
eve       Mon Jun 24 10:25:20 2019  58.574
frank     Mon Jun 24 10:25:20 2019  85.362
```

The parameterized query lets the administrator run the same query for specific accounts.

If Slurm accounts for physics `phys` and mathematics `math` have been created with

```
[root@basecm11 ~]# sacctmgr add account phys,math
```

then `account=phys` and `account=math`, are the <*key*>="<*value*>" format options. The query can then be run with the `-p|--parameter` option as follows:

```
[basecm11->monitoring->labeledentity]% instantquery -q users_job_wall_clock_per_account_second\
s -p account=phys -p period=1w
user      Timestamp                 Value
--------- ------------------------- --------
alice     Mon Jun 24 10:25:22 2019  28.787
```

```
charline  Mon Jun 24 10:25:22 2019  29.787
frank     Mon Jun 24 10:25:22 2019  30.788
[basecm11->monitoring->labeledentity]% instantquery -q users_job_wall_clock_per_account_second\
s -p account=math -p period=1w
user      Timestamp                 Value
--------- ------------------------- --------
bob       Mon Jun 24 10:25:37 2019  26.787
charline  Mon Jun 24 10:25:37 2019  73.049
eve       Mon Jun 24 10:25:37 2019  30.787
```

12.4.1 Two Job GPU Metrics Used In PromQL Queries

There are two important job GPU metrics (section G.1.8) that are used in several PromQL queries. These two are:

1. The `job_gpu_utilization` GPU metric:

   This is based on the `gpu_utilization` metric collected on GPU nodes via the DCGM library. The values it takes are in the interval [0, 1]. In DCGM the name of the metric is `DCGM_FI_DEV_GPU_UTIL`, and it represents the total GPU utilization.

   The `gpu_utilization` metric values collected on the GPU are mapped to the job that uses the GPU at the time of the collection.

   `job_gpu_utilization` is a labeled entity, tagged with a label representing a job. This allows the person carrying out PromQL queries to use such parameters as `job id`, `job user`, and so on. It is assumed that one GPU is used only by processes of a single job, but there can be several GPUs, each used by different jobs simultaneously. Each of the GPUs on the node has independent values of `job_gpu_utilization`.

   One example of a PromQL query that uses the `job_gpu_utilization` GPU metric is `job_names_usage_gpu`, which has the query expansion:

   ```
   sum by(job_name) (
     sum_over_time(job_gpu_utilization[${period}])
   )
   ```

2. The `job_gpu_wasted` GPU metric:

   The `job_gpu_wasted` metric is a labeled entity (tagged metric) and is based on the `job_gpu_utilization` metric. The `job_gpu_wasted` metric shows what fraction, out of 1, of the GPUs on the node were unused by a job despite being allocated by the workload manager for that job. It can take values in the interval [0, 1].

   It is calculated as follows:

   $1 - \frac{\texttt{all\_gpus\_utilization}}{\texttt{requested\_gpus}}$

   where

   - `all_gpus_utilization` is the average utilization for all `gpu_utilization` metric values collected in the interval [-1; +1] from the time of the metric calculation for all GPUs requested by the job on the node
   - `requested_gpus` is the number of GPUs allocated for the job in the workload manager

   For example:

   If a job requests two GPUs on a node, and the job does not use any of those GPUs at all, then `job_gpu_wasted` takes the value 1.

If the same job uses half of the first GPU and does not use the second one, then the metric value is calculated as:

$1 - \frac{(0.5+0)}{2} = 0.75$

One example of a PromQL query that uses `job_gpu_wasted` is `jobs_wasted_allocated_gpus`, which has the query expansion:

```
avg by(job_id) (
  round(100 * avg_over_time(job_gpu_wasted[${period}]))
)
```

12.5 Job Accounting In Base View

Job accounting in Base View is designed to present accounts of jobs without having to construct command lines with syntaxes that can be tricky to deal with. One useful output format is as basic Excel-format spreadsheets.

Job accounting can be viewed within Base View's accounting mode available using the top right hand side of the Base View standard display (figure 10.5).

By default, job accounting opens up with a dashboard panel called `K8S Container Metrics`, which provides Kubernetes container metrics widgets (figure 12.1):

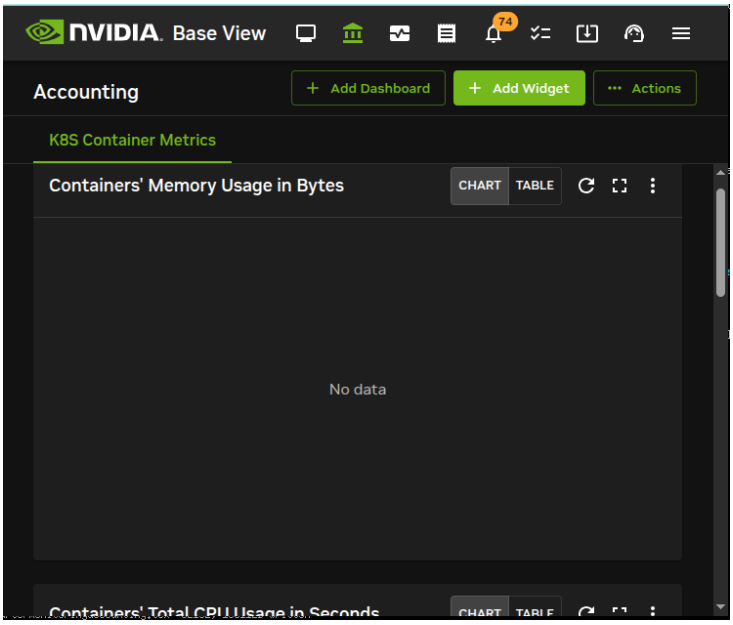


Figure 12.1: Job accounting: K8S container metrics panels

If Kubernetes containers have been running jobs that have been sampled, then pre-selected container-related queries can be run from these widgets, and the data samples can be displayed as tables or plots. If Kubernetes containers have not been running jobs, then the pre-selected container-related queries have no data samples to display (figure 12.1).

- By default, the dashboard panel has some pre-selected queries related to user jobs over the last period.
  - `Containers' Memory Usage in Bytes`
  - `Containers' Total CPU Usage in Bytes`
  - `Containers' Total Received Bytes`

– Containers' Total Filesystem Usage in Bytes

The default layout requires scrolling down to view the other queries. Other layouts can be selected from the ··· Actions > Edit Dashboard menu option (figure 12.1).

The dashboard can be managed generally using the same ··· Actions button, to also reach menu options to delete, clone, reset, export, and import dashboards.

Clicking the + Add Dashboard button next to the Accounting header creates an entirely new dashboard panel within accounting.

- Clicking the +Add Widget button for a BCM dashboard panel, for example the K8S Container Metrics panel in figure 12.1, creates a new widget in the dashboard panel.
  - A query can be selected for the widget by right-clicking on the collapsed menu associated with it, using the navigation path : ⋮ > ⚙ (figure 12.2):

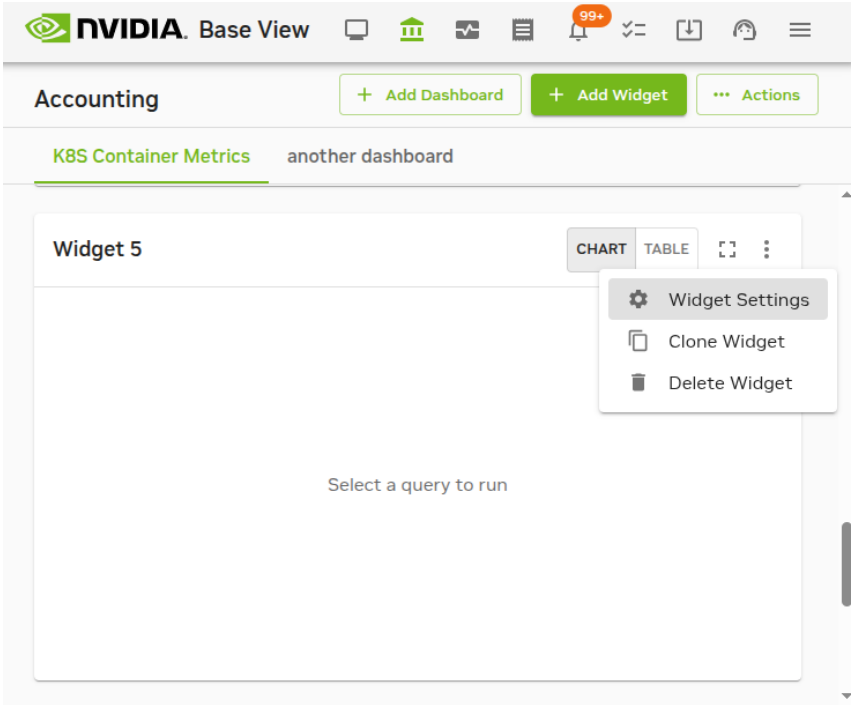


Figure 12.2: Job accounting: widget settings selected from ⋮ menu

A query can be either be a selected query, using the SELECT QUERY tab, or it can be a constructed query, using the PLAIN TEXT QUERY tab:

- For a selected query (figure 12.3), the Query and Query preview box are initially blank. A dropdown menu allows a query to be selected and saved. In figure 12.3 the selected query is gpu_power_usage_for_account:

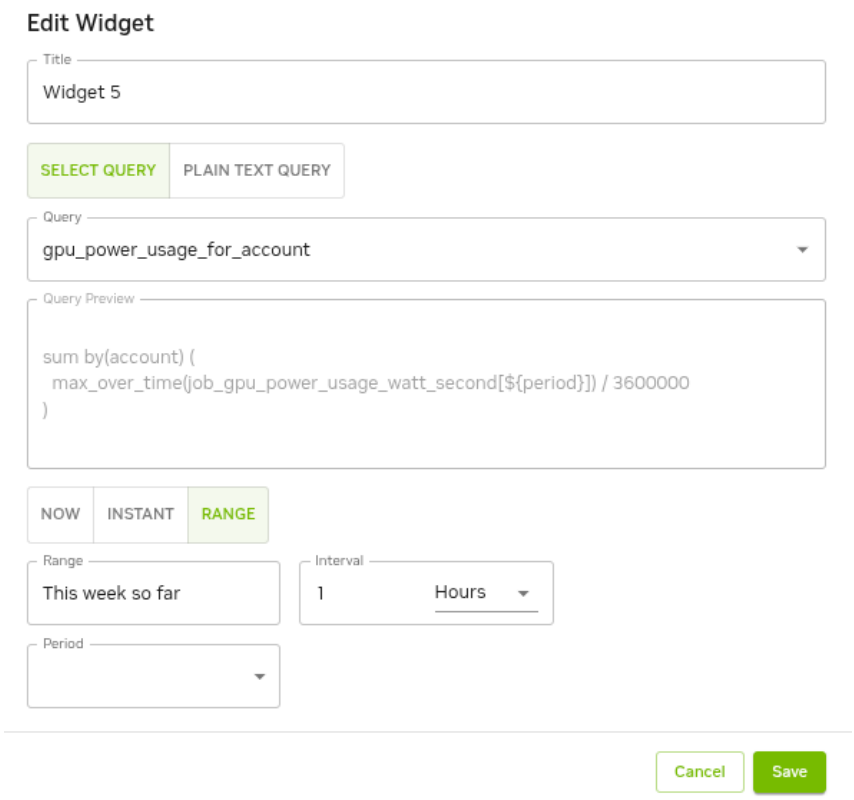


Figure 12.3: Job accounting: query selection for widget

- For a constructed query (figure 12.4), the text can be constructed in the `Query` box and saved:

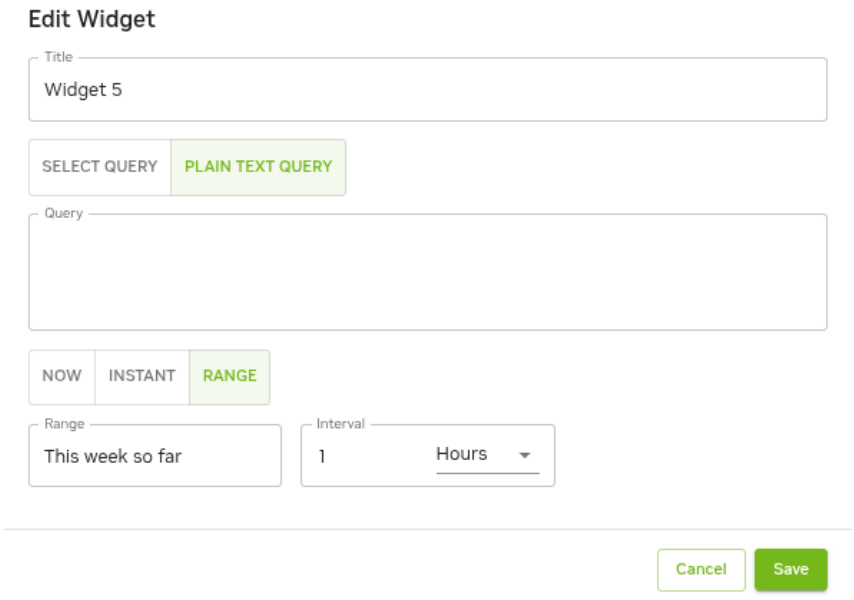


Figure 12.4: Job accounting: query construction for widget

12.5.1 Management And Use Of The Accounting Panel

The following description is illustrated by figure 12.1:

A Base View job accounting dashboard is made up of collections of Base View job accounting panels, also known as widgets.

- To add a new dashboard, the `+ Add Dashboard` button next to the `Accounting` header can be clicked. Other dashboard management actions can be carried out with the `··· Actions` menu options.
- To add a new panel, the `+ Add Widget` button next to the `Accounting` header can be clicked. Other widget management actions can be carried out with the ⋮ menu options.

Job accounting is intended to present accounts of jobs. So, when a new panel is created, the ⋮ > `Widget Settings` menu option (figure 12.2) displays a dialog that asks for inputs such as the range in which the report is to be carried out, the sampling interval, and the report name (figure 12.3).

If the dialog has `SELECT QUERY` set (figure 12.3), then PromQL queries can be selected from a drop-down menu in the `Query` box (figure 12.5):

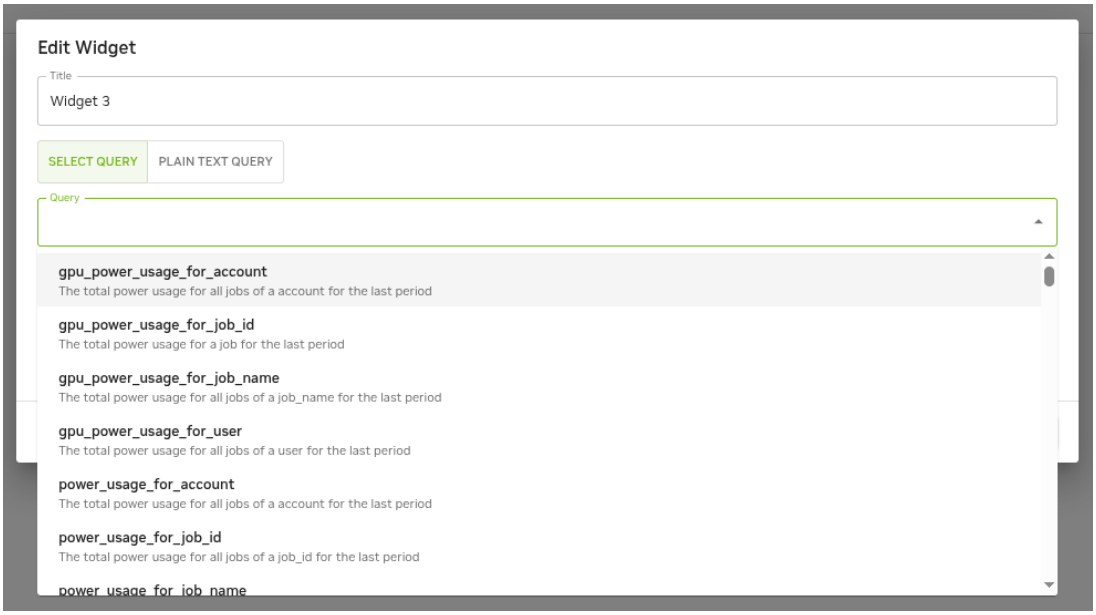


Figure 12.5: Job accounting: query selection for widget

By default, when the new dashboard is saved, the selected queries are run and a table can be seen of the results.

The results of a run can be displayed according to the widget settings options of `NOW`, `INSTANT`, and `RANGE` (these options can be seen in figures 12.3 and 12.4).

- Selecting `NOW` lets the output be displayed for the current time, as a pie chart (figure 12.6):

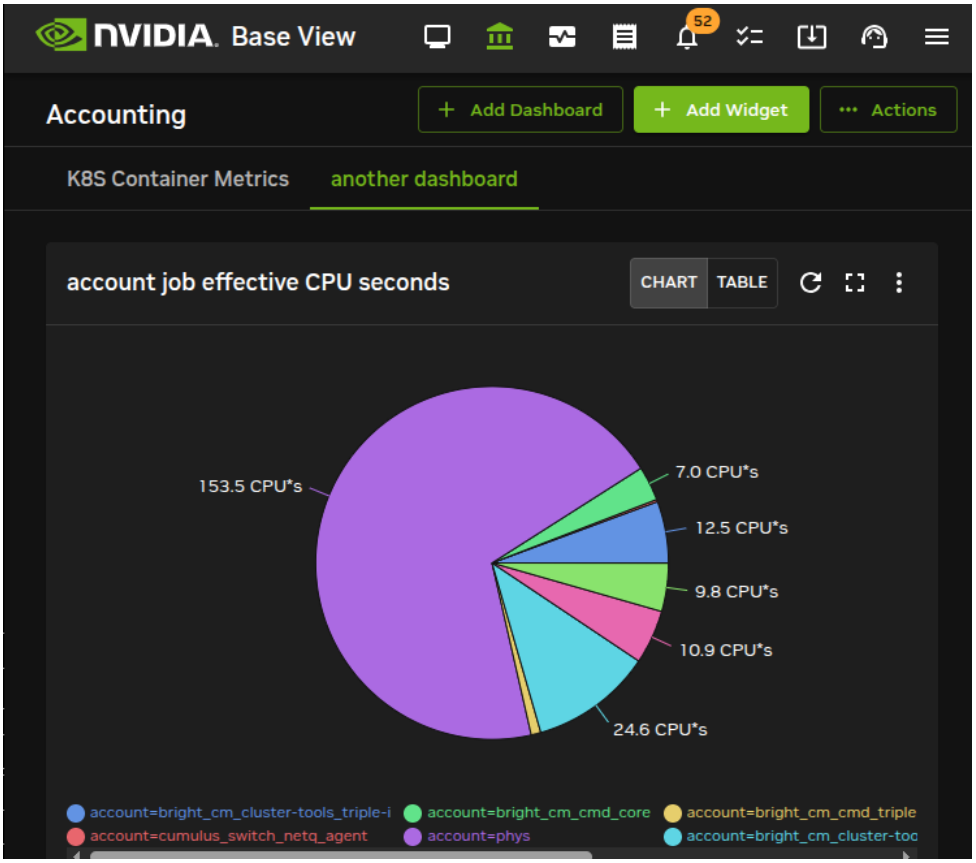


Figure 12.6: Job accounting: PromQL query pie chart display for the current time

or as a table (figure 12.7):

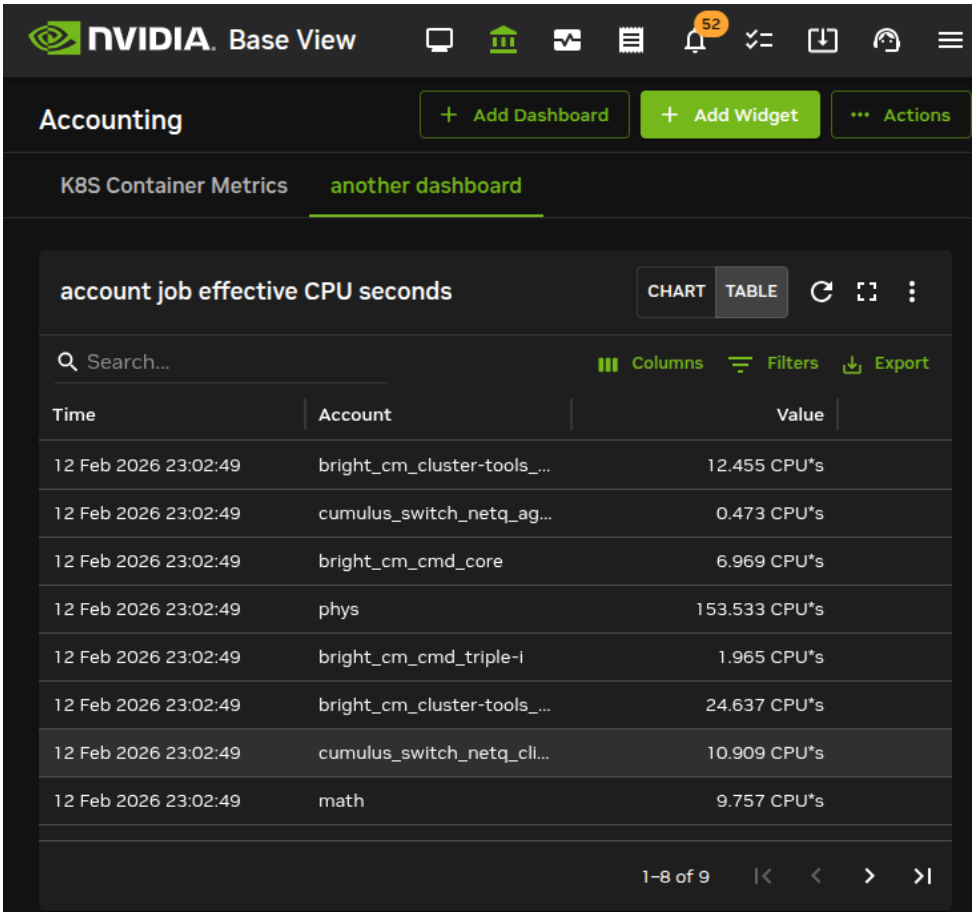


Figure 12.7: Job accounting: PromQL query table display for the current time

- Selecting `INSTANT` lets the output be displayed as a pie chart or as a table too, but for a specified time in the past.
- Selecting `RANGE` lets the output be displayed for a specified time range. This can be as a graph (figure 12.8)

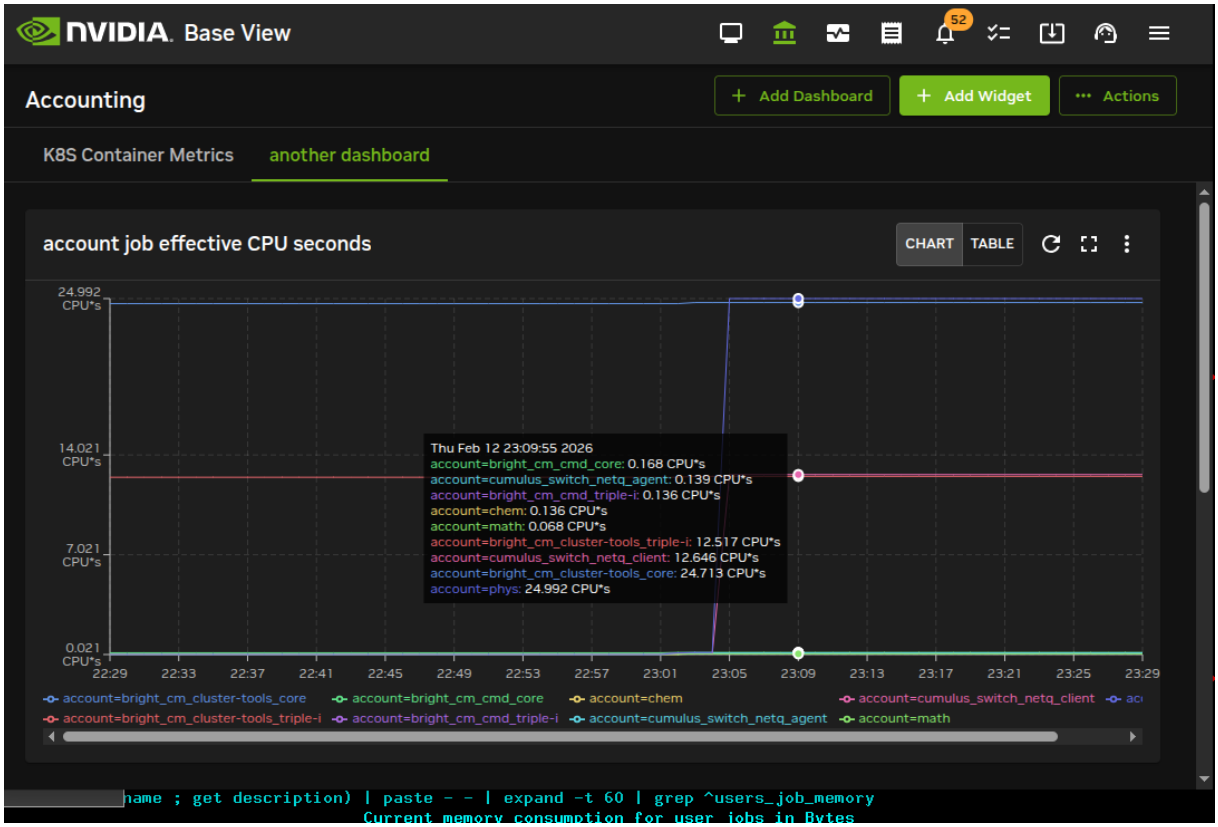


Figure 12.8: Job accounting: PromQL query range graph display for a specified time range

or as a table (figure 12.9)

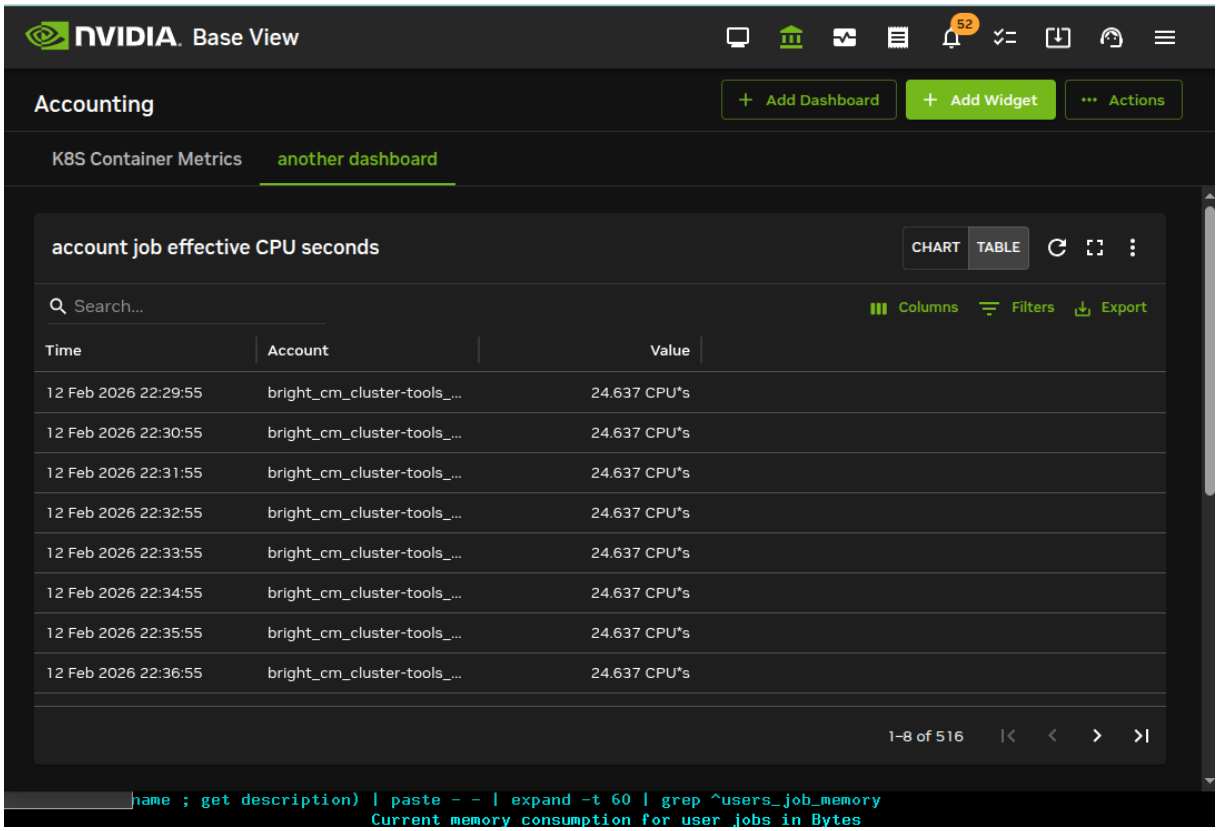


Figure 12.9: Job accounting: PromQL query range table display for a specified time range

For each option, the table display can be downloaded as a CSV file by clicking on `Export` which is convenient for plotting custom graphs in forms other than what Base View offers.

The Range Query Type

The range query is accessible in Base View by clicking on the `RANGE` option in figures 12.3 or 12.4. The time range over which the query is executed can then be set. On saving, a value is fetched for each interval within that time range, and plotted. Very long ranges can be computationally expensive.

The query submitter can set absolute `Start time` and `End time` parameters using the Base View navigation path (figure 12.10)

`Accounting > ⋮ > Widget Settings > RANGE > Range > ABSOLUTE MODE`

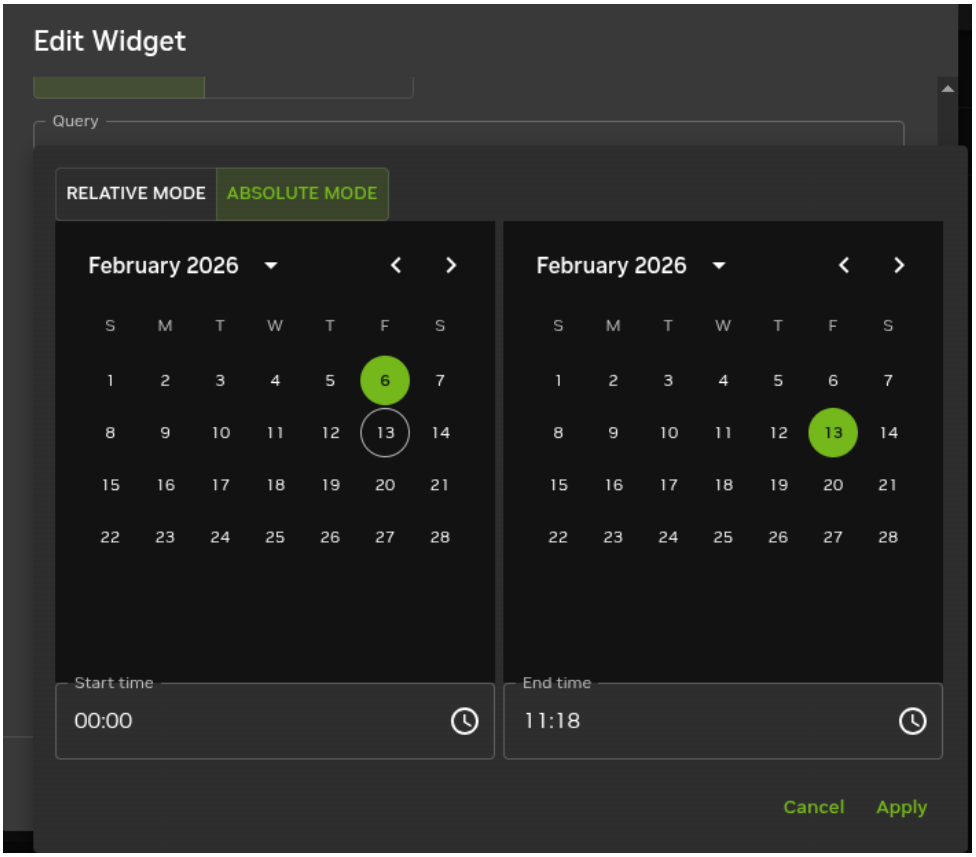


Figure 12.10: Job accounting: setting a range start time and end time

The query `users_job_memory_usage_bytes` is an example of the range query type. It fetches the total memory usage for jobs over intervals, grouped per user during that interval (figure 12.11).

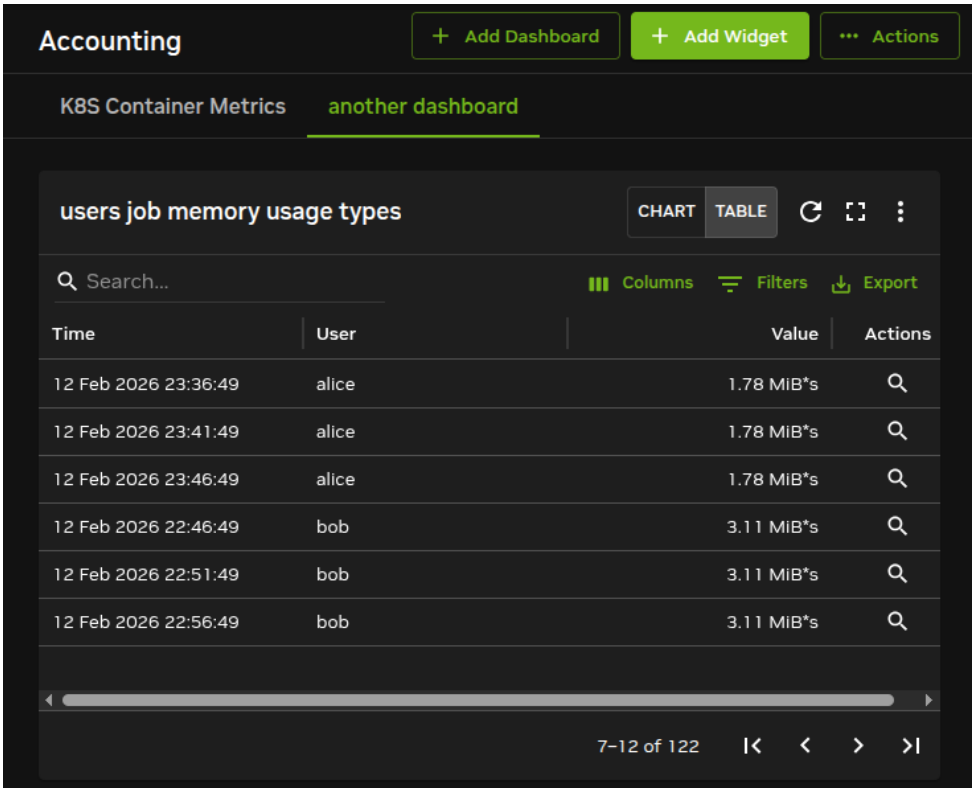


Figure 12.11: Job Accounting: User Values Over A Range

The Interval Setting

The interval is the plot interval along the time axis. When editing a range query type, the value set in the interval field for a range query run is set when the `Save` button is clicked. Lower numbers lead to smaller intervals, hence higher resolution and higher query execution times. For `interval` in `cmsh`, a value of `0` means that a reasonable default interval is attempted. Setting `0` is not valid in Base View plots.

Choosing an interval that is less than the sampling time of the metric is a bit pointless, and tends to lead to data values that display non-smooth behaviour.

12.6 Access Control For Workload Accounting And Reporting

The ability to view jobs is controlled by four tokens defined in a user's profile:

1. `GET_JOB_TOKEN`: Allows all running jobs to be seen.
2. `GET_OWN_JOB_TOKEN`: Allows owned running jobs to be seen.
3. `GET_JOBINFO_TOKEN`: Allows all cached historic and running jobs to be seen.
4. `GET_OWN_JOBINFO_TOKEN`: Allows owned cached historic and running jobs to be seen.

To retrieve monitoring data for the job, the token `PLOT_TOKEN` must also be defined for the profile.

A job is always owned by the user that runs it. Ownership of a job can also be shared with other users by defining *project managers*. This establishes a 2-level hierarchy, with project managers above the *subordinates*, who are users that are assigned to the project manager. One or more *accounts*, can be assigned to project managers.

12.6.1 Defining Project Managers Using Internal User Management

Any user can be turned into a project manager. If the BCM LDAP server is being used, then the project manager can be configured via `cmsh`.

Example

alice can be made the project manager of bob and charlie. This allows her access to the job data of her subordinates:

```
[basecm11->user]% projectmanager alice
[basecm11->user*[alice*]->projectmanager*]% set users bob charlie
[basecm11->user*[alice*]->projectmanager*]% commit
```

Example

albert can be made the manager of the physics account. This gives him access to all jobs running under that account:

```
[basecm11->user]% projectmanager albert
[basecm11->user[albert*]->projectmanager*]% set accounts physics
[basecm11->user[albert*]->projectmanager*]% commit
```

Both mechanisms, users and accounts, can be combined to provide access control.

Example

To limit access to jobs that are running under the physics account to a specific set of users:

```
[basecm11->user]% projectmanager albert
[basecm11->user*[albert*]->projectmanager*]% set accounts physics
[basecm11->user*[albert*]->projectmanager*]% set users niels richard
[basecm11->user*[albert*]->projectmanager*]% set operator and
[basecm11->user*[albert*]->projectmanager*]% commit
```

12.6.2 Defining Project Managers Using External User Management

If an external user management server is used instead of the BCM LDAP server, then project managers cannot be defined in cmsh. Instead, a script has to be written that provides the definitions for project managers in the form of a JSON object.

The full path of the script, for example /path/to/the/script, has to be set as a value to the ProjectManagerScript parameter in cmd.conf. This is done by adding it to the AdvancedConfig directive (page 925):

Example

```
AdvancedConfig = { "ProjectManagerScript=/path/to/the/script" }
```

The directive becomes active after restarting CMDaemon.

An example of a project manager script can be found at /cm/local/apps/cmd/scripts/cm-project-managers.py. It gives users access to each other's jobs if they share at least one group.

The easiest way to use the script is use a mapping to inform CMDaemon which of the other user's jobs each user has access to.

Example

So if frank can access data belonging to bob and dennis, while bob can access data belonging to dennis, while dennis can only access his own data, then the project manager configurations can be set up as:

```
{
  "frank": ["bob", "dennis"],
  "bob": ["dennis"],
  "dennis": []
}
```

Account access control can also be included in the output of the script, by setting values for the `users`, `accounts`, and the boolean operator (`and`, `or`) options:

Example

```
{
  "alice": {
    "users": ["charline", "eve"],
    "accounts": ["math, "chem"],
    "and": True
  },
  "eve": {
    "accounts": ["chem"]
  }
}
```

After restarting CMDaemon, it automatically runs this script when committing a change for a device or data producer. It is also possible to manually trigger the script to be run on the active head, by executing:

```
[root@basecm11 ~]# echo "PROJECT.MANAGERS.UPDATE" > /var/spool/cmd/eventbucket
```

The script must not take more than a few seconds to process.

Workaround For Project Manager Script That Takes Too Long

If the script takes longer to run, then it must be run outside of CMDaemon, and its output should be saved as a file. If the output file is located at `/path/to/the/file`, then its path can be set as an input to the `ProjectManagerFile` parameter in `cmd.conf`. This is done by adding it to the `AdvancedConfig` directive (page 925):

Example

```
AdvancedConfig = { "ProjectManagerFile=/path/to/the/file" }
```

The project manager definitions become active after CMDaemon is restarted.

12.7 Drilldown Queries For Workload Accounting And Reporting

Metrics can be classified in various ways. Common ways are by:

- device: A typical hardware device is a node. Each node can then have its metrics, which are CPU usage, memory, storage, and other resource use, displayed over time. Device metrics are largely covered in this chapter in the sections up to and including section 10.8.
- job: With this classifier, each job that is run by a workload manager can have its metrics, which are CPU usage, memory, storage and other resource use, displayed over time. Job metrics are covered in Chapter 11.

Other ways of classifying metrics are part of workload accounting. With workload accounting, a workload manager runs jobs, and a job metric can be classified by:

- user
- job (job ID)
- account
- job name

The classification can be carried out singly. However, it can also be carried out at the same time, like filters. For example:

- each user could be classified for a particular job metric
  or
- a particular user could be classified for a particular job metric
  or
- a particular user could be classifed for a particular job metric for a particular job ID only
  or
- a particular user could be classified for a particular job metric for that particular job ID only for a particular account only

A cluster administrator that uses several filters to get to the "bottom" of how resources are being used, functions in a manner reminiscent of someone drilling to the bottom to find something. This type of filtering is therefore called *drilldown*. Each filter corresponds to a *level* of drilldown.

Drilldown is a bit like how in `cmsh` the use of the `filter` command within `jobs` mode can narrow down what is displayed:

Example

```
[basecm11->wlm[slurm]->jobs]% filter -n iozone -u edgar -a projecty
Job ID  Job name  User   Queue  Submit time    Start time  End time   Nodes   Exit code
------  --------  ------ -----  -----------    ----------  --------   -----   ---------
15      iozone    edgar  defq   14:22:05       14:50:22    15:00:24   node001 0
19      iozone    edgar  defq   14:26:07       15:10:25    15:20:19   node001 0
25      iozone    edgar  defq   14:34:57       15:30:20    15:40:39   node001 0
36      iozone    edgar  defq   14:43:54       16:10:15    16:19:57   node001 0
41      iozone    edgar  defq   14:47:24       16:19:57    16:29:42   node001 0
```

except that in `cmsh` the value of the job metric is not specified or shown by the `filter` command.

Drilldown is also rather similar in concept to how pivot tables are used in Excel spreadsheets. Pivot tables are particular selections (like the filtered choices in drilldown). The selections are applied to a great deal of raw data. A function (like the metric in drilldown) is applied to the selection, to present the information more clearly to the end viewer.

In contrast with the filter output of `cmsh`, in Base View the job metric value is made visible in a table or in graphs over the period.

12.7.1 The `drilldownoverview` Command

The list of predefined drilldown queries of section 12.3.1 can be listed with the `drilldownoverview` command. This can be run from the `query` submode of the `monitoring` mode. The output, with some columns cut out for convenience, looks like:

Example

```
[basecm11->monitoring->query]% drilldownoverview |cut -b1-54,80-143

Query                                                Drill down query
---------------------------------------------------- ----------------------------------------------------
accounts_wasted_memory                               wasted_memory_job_name_for_account (1)
job_information_by_account (0)                       job_information_by_job_name_for_account (1)
job_information_by_account (0)                       job_information_by_user_for_account (1)
job_information_by_job_name_for_account (1)          job_information_by_user_for_account_and_job_name (2)
job_information_by_job_name_for_account_and_user (2) job_information_by_job_id_for_account_and_user_and_job\
```

```
                                                                                _name (3)
job_information_by_job_name_for_user (1)                job_information_by_job_id_for_user_and_job_name (2)
job_information_by_user (0)                             job_information_by_job_id_for_user (1)
job_information_by_user (0)                             job_information_by_job_name_for_user (1)
job_information_by_user_for_account (1)                 job_information_by_job_name_for_account_and_user (2)
job_information_by_user_for_account_and_job_name (2)    job_information_by_job_id_for_account_and_user_and_job\
                                                                                _name (3)
users_job_effective_cpu_seconds                         job_effective_cpu_seconds_job_name_for_user (1)
users_job_io_bytes                                      job_io_bytes_per_job_name_for_user (1)
users_job_memory_usage_bytes                            job_memory_usage_bytes_per_job_name_for_user (1)
users_job_running_count                                 job_running_count_job_name_for_user (1)
users_job_waiting_seconds                               job_waiting_seconds_job_name_for_user (1)
users_job_wall_clock_seconds                            job_wall_clock_seconds_job_name_for_user (1)
users_unused_gpu                                        unused_gpu_job_name_for_user (1)
users_used_gpu                                          used_gpu_job_name_for_user (1)
users_wasted_allocated_gpus                             wasted_allocated_gpus_for_user (1)
users_wasted_memory                                     wasted_memory_job_name_for_user (1)

[basecm11->monitoring->query]%
```

The queries have drilldown options. For example, the `job_information_by_account`:

```
[basecm11->monitoring->query]% use job_information_by_account
[basecm11->monitoring->query[job_information_by_account]]% show
Parameter                      Value
------------------------------ ------------------------------------------------------------
Name                           job_information_by_account
Revision
Class                          drilldown/level/0
Alias
Start time                     now
End time
Interval                       0s
Description                    Generic job information drill down query grouped by account
PromQL Query                   <61B>
Access                         Public
Unit
Price                          0.000000
Currency                       $
Preference                     0
Drill down                     <2 in submode>
Notes                          <0B>
[basecm11->monitoring->query[job_information_by_account]]%
[basecm11->monitoring->query[job_information_by_account]]% drilldown
[basecm11->monitoring->query[job_information_by_account]->drilldown]% list
Name (key)   Parameters   Query
------------ ------------ ---------------------------------------
job_name     account      job_information_by_job_name_for_account
user         account      job_information_by_user_for_account
[basecm11->monitoring->query[job_information_by_account]->drilldown]% use job_name
[basecm11->monitoring->query[job_information_by_account]->drilldown[job_name]]% show
Parameter                      Value
------------------------------ ------------------------------------------------
Name                           job_name
Revision
Parameters                     account
Query                          job_information_by_job_name_for_account
```

12.8 The grid Command For Job Accounting

A way to carry out job accounting without relying on PromQL queries is the grid command of cmsh. The grid command can be accessed from within wlm mode for a workload manager. The command displays the nodes in a grid.

An example of grid output can be seen in the session of page 716.

As seen in that output, the nodes are displayed in a sequence of rows. For each row, the following row of nodes has the same sequence of nodes, but after an interval of time. Each node in the grid is displayed as a colored block called a *timeblock*. The timeblock displays a value for the classifier entity that the grid command associates with that node. The value for the timeblock is indicated by either its color, or by text superimposed on that timeblock, or both.

12.8.1 The grid Command Help Text

The command options to grid can be looked up in the help text:

Example

```
[basecm11->wlm[slurm]]% help grid
Name:
          grid - Show a grid of historic job information

Usage:
          grid [options]

Options:
          -n, --nodes node(list)
               List of nodes, e.g. node001..node015,node020..node028,node030 or ^/some/file/containing/hostnames

...
```

The classifier entity shown by grid can be called its *mode*, and it can be set with the --mode option. By default the mode is set to used, which means the classifier entity value displayed for the node is either that it is being used to run a job, or not being used to run a job.

A help text to describe modes available for a timeblock can be seen with:

Example

```
[basecm11->wlm[slurm]]% grid --mode x
Mode x is not implemented
Valid modes:

used:         the percentage of the time the node was used in this timeblock (default)
count:        the number of jobs using the node in this timeblock
average:      the averaged time jobs used the node in this timeblock
user:         the user that used the node the most in this timeblock
group:        the group that used the node the most in this timeblock
account:      the account that used the node the most in this timeblock
job-name:     the job-name that used the node the most in this timeblock
job-id:       the job-name that used the node the most in this timeblock
count-cpu:    the number of requested CPUs on the node in this timeblock
average-cpu: the time averaged of requested CPUs on the node in this timeblock
count-gpu:    the number of requested GPUs on the node in this timeblock
average-gpu: the time averaged of requested GPUs on the node in this timeblock
```

12.8.2 Some grid Command Examples

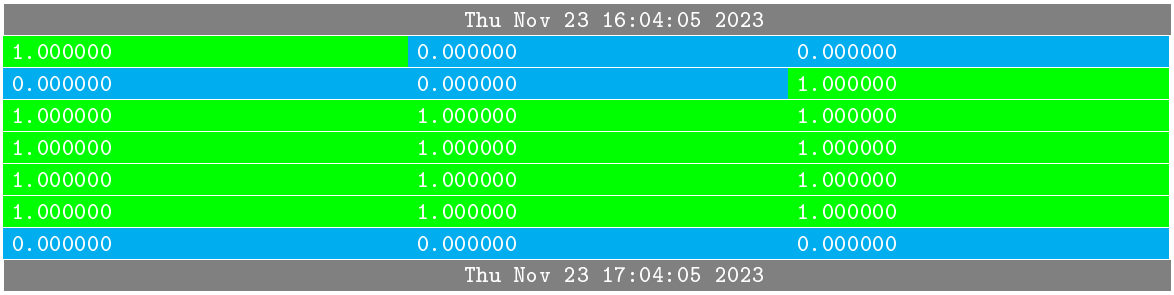
With the grid command, the timeblock value is given a color, and can also be indicated by an associated text value.

Displaying Nodes Used

For example: over intervals of 10 minutes (600 seconds), from 28 hours ago to 27 hours ago, for nodes `node001`, `node002`, `node003`, the `used` mode for nodes can be displayed with the command:

Example

```
[basecm11->wlm[slurm]]% grid --start -28h --end -27h --interval 600 -n node001..node003 --text --legend
node001                          node002                          node003
```





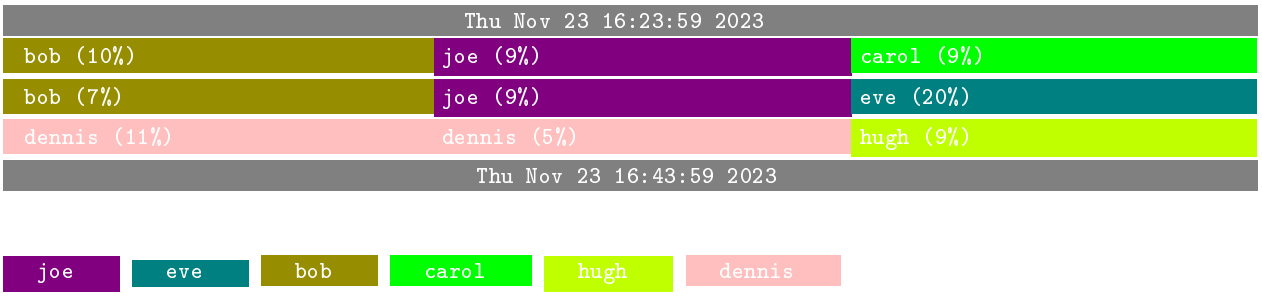
The `--legend` option provides a color legend after the display, and the `--text` option for this mode overwrites the timeblock with an associated value of `1.00000` or `0.000000`.

The time specification is explained in detail in section 12.8.3.

Displaying Users Usage Of Nodes

Another example, which displays users that used the node the most, and their usage as a percentage, per timeblock of 600s, over a time from 5370 minutes in the past to 5350 minutes in the past:

```
[basecm11->wlm[slurm]]% grid --mode user --start -5370m --end -5350m -n node001..node003 --interval 600 \
                  --text --legend
 node001                          node002                          node003
```





The display percentage can be disabled with the `--no-percent` option.

Displaying Larger Numbers Of Nodes

For larger clusters, the administrator can view the grid output over a larger monitor, or over several monitors, and perhaps using higher resolutions to get a colour map. Figure 12.12 shows an output for 128 nodes:

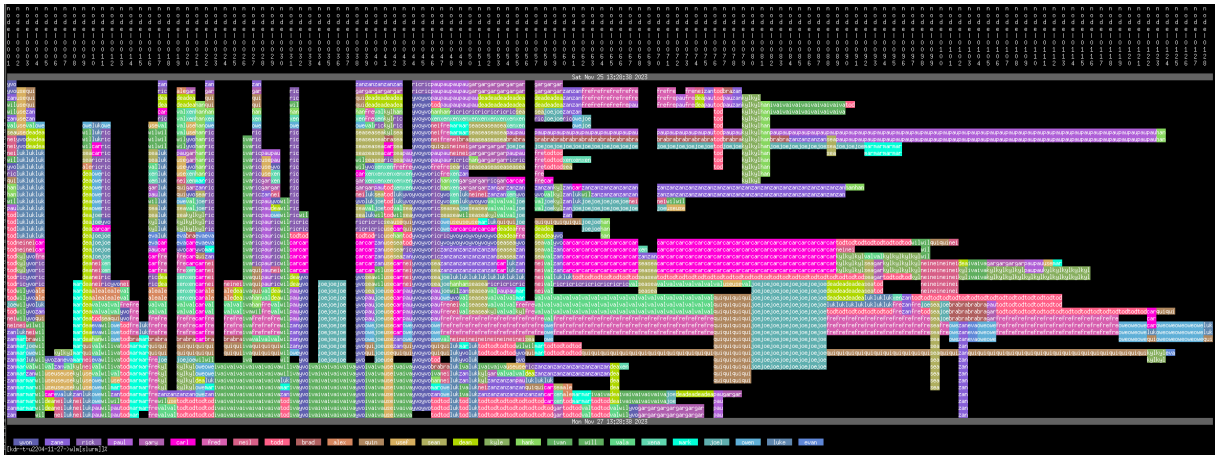


Figure 12.12: Job Accounting: Grid account output for 128 nodes

The watch command of cmsh can be used to display updates as the cluster is used during the day, without requiring input from the cluster administrator. Patterns of use may be viewed in this manner.

For example, the black parts of node rows in figure 12.12 are a visual indication of times when those nodes were not used.

12.8.3 The grid Command Time Specification

The grid command time specification is identical to that of the dumpmonitoringdata command (section 10.6.4), except that dumpmonitoringdata has implicit mandatory arguments. The time specifications for grid are, on the other hand, done explicitly, and use the time options and arguments:

- --end <*end-time*>

  and

- --start <*start-time*>

The syntax for the time specifications of grid is also used by the statistics and filter commands. The time options --start and --end can have their arguments specified as follows:

- *Fixed time format*: The format for the times that make up the time pair can be:
  - [YY/MM/DD] HH:MM[:SS]
    (If YY/MM/DD is used, then each time must be enclosed in double quotes)
  - The unix epoch time (seconds since 00:00:00 1 January 1970)
- now: For the --end option, a value of now can be set. The time at which the grid command is run is then used.
- *Relative time format*: One item in the time pair can be set to a fixed time format. The other item in the time pair can then have its time set relative to the fixed time item. The format for the non-fixed time item (the relative time item) can then be specified as follows:
  - For the <*start-time*>, a number prefixed with "-" is used. It indicates a time that much earlier than the fixed end time.
  - For the <*end-time*>, a number prefixed with "+" is used. It indicates a time that much later than the fixed start time.
  - The number values also have suffix values indicating the units of time, as seconds (s), minutes (m), hours (h), or days (d).

The relative time format is summarized in the following table:

Unit	<*start-time*>	<*end-time*>
seconds:	-<*number*>s	+<*number*>s
minutes:	-<*number*>m	+<*number*>m
hours:	-<*number*>h	+<*number*>h
days:	-<*number*>d	+<*number*>d

- Both <*start-time*> and <*end-time*> can have their values prefixed with a "-". In this case, the range over which the monitored values are seen is in the past, relative to the current time. If the end time for the range is specified as further in the past than the starting time, then the time values are swapped over so that the end time becomes more recent than the starting time.

13

Monitoring: Job Chargeback

13.1 Introduction

13.1.1 The Word "Chargeback"

In a non-IT context, the term chargeback is commonly used with credit cards, in a situation where a cardholder disputes a payment that was made to a merchant. When the credit card company pays back the disputed payment to the card holder, that is called a chargeback.

In an IT context, the term chargeback still has to do with money, and the idea of getting money back. However, in practice the intention ideally is not about disputing costs, but rather about measuring what the costs are of the IT resources that have been requested. Measurement of requested resources means that there is a potential to charge back the users or groups of users who requested these resources.

So, for example, an IT department for an organization may be allocated a budget to run a cluster, meant for the benefit of the organization. The IT department may make the cluster available to many other departments. These other departments request cluster resources. The IT department measures the requested resources and charges back the associated costs.

If a cluster is used by several different departments in the organization, then a simple way to pay for resource requests is to spread the entire cost as a general overhead expense over all departments equally. That may make matters easy for the cluster administrator, but can be harmful to the organization, because without a fair resource request management, there is a tendency for resource request abuse.

If however the resource requests per department are measured, it means that department managers can be kept aware of how resource requests are being divided up. Being able to measure requested resources per department, and thus being able to charge back the department for the requested resources, means that the organization using the cluster can plan and manage resource request budgets efficiently and fairly.

13.1.2 Comparison Of Job Chargeback Monitoring Measurement With Other Monitoring Measurements

Monitoring measurements in BCM in general can be considered to be:

- Monitoring of devices (Chapter 10), which is about monitoring devices in a cluster.
- Job monitoring (Chapter 11), which is about using monitoring to measure the resources used by jobs that actually run.
- Job accounting (Chapter 12), which is about using monitoring to measure the resources used by a user, a group, or other classifier entity.
- Job chargeback (this chapter), which is about using monitoring to measure the resources requested for a job, whether the resources are used or not. This allows the requester to be charged the costs of requesting those resources.

The difference between actual resource use and requested resource use can be illustrated by a thought experiment:

First, a job is run that runs the CPU at 100% for 10 minutes. After that job is completed, a second job is run that runs a `sleep` command that lasts 10 minutes.

If these two jobs are considered from the point of view of CPU resource usage, then according to the workload manager:

The first job actually uses the CPU for 10 minutes, and the administrator can work out from the job monitoring system what user that ran that job, and charge the user for CPU usage.

For the second job, the administrator cannot use the job monitoring system to charge the user for the CPU usage because there was no significant usage measured. However, resources were used up during this time, because the request for the job blocked the availability of that CPU during this period for other users. The administrator would like to charge the user for preventing others from accessing the resources during that period. To do that requires measuring the resources requested during that period, rather than resources that were really used.

Thus, the aim of job chargeback monitoring is to provide a way to track resource requests, which typically differs a little from resource usage.

13.2 Job Chargeback Measurement

13.2.1 Predefined Job Chargebacks

Some predefined chargebacks can be listed and configured under the `chargeback` submode of the `wlm` mode.

The predefined list is:

```
[basecm11->wlm[slurm]]% chargeback
[basecm11->wlm[slurm]->chargeback]% list
                                                    Group by Group by  Group by Price per  Price per
Name (key)                                          user     group     account  CPU second GPU second
--------------------------------------------------- -------- --------  -------- ---------- ----------
Jobs completed last month grouped by account        no       no        yes      8.64 $/d   1.80 $/h
Jobs completed last month grouped by group          no       yes       no       8.64 $/d   1.80 $/h
Jobs completed last month grouped by user           yes      no        no       8.64 $/d   1.80 $/h
Jobs completed last month grouped by user and account yes    no        yes      8.64 $/d   1.80 $/h
Jobs completed this month grouped by account        no       no        yes      8.64 $/d   1.80 $/h
Jobs completed this month grouped by group          no       yes       no       8.64 $/d   1.80 $/h
Jobs completed this month grouped by user           yes      no        no       8.64 $/d   1.80 $/h
Jobs completed this month grouped by user and account yes    no        yes      8.64 $/d   1.80 $/h
Jobs completed this year grouped by account         no       no        yes      8.64 $/d   1.80 $/h
Jobs completed this year grouped by group           no       yes       no       8.64 $/d   1.80 $/h
Jobs completed this year grouped by user            yes      no        no       8.64 $/d   1.80 $/h
Jobs completed this year grouped by user and account yes     no        yes      8.64 $/d   1.80 $/h
```

Setting A Price For Resource Requests

The pricing can be set per resource requests over a specified time period. In the preceding list, the period is a month or a year. The resource and associated resource request consumption can be the following pairs:

Resource And Resource Request Consumption Pairs	
Resource	**Resource Request Consumption**
CPU	CPU second
GPU	GPU second
CPU core	CPU core second
slot	slot second
memory bytes	byte-second
Power	Watt-second (Joules)

Different workload managers use different resources for resource measurement, which is why there is a variety in the resources that can be used for pricing.

13.2.2 Setting A Custom Job Chargeback

In addition to the predefined job chargebacks, more chargebacks can be added. For example, the number of jobs completed so far today can be set up as follows:

Example

```
[basecm11->wlm[slurm]->chargeback]% add "Jobs completed this day grouped by user"
[basecm11->...ck*[Jobs completed this day grouped by user*]]% show
Parameter                      Value
------------------------------ ------------------------------------------------
Name                           jobs completed this day grouped by user
Revision
Notes                          <0B>
Group by user                  no
Group by group                 no
Group by account               no
Group by job name              no
Group by job ID                no
Group by accounting info
Users
Groups
Accounts
Job names
Job IDs
Accounting info                < not set >
Price per CPU second           0.00 $/s
Price per CPU core second      0.00 $/s
Price per GPU second           0.00 $/s
Price per memory byte-second   0$/B*s
Price per slot second          0$/slot*s
Price per Watt second          0$/W*s
Currency                       $
Start time                     now/M
End time                       now/M
UTC                            no
Include running                no
Calculate prediction           no
Preference                     0
```

```
[basecm11->...ck*[Jobs completed this day grouped by user*]]% set pricepercpusecond 0.0001$/s
[basecm11->...ck*[Jobs completed this day grouped by user*]]% set groupbyuser yes
[basecm11->...ck*[Jobs completed this day grouped by user*]]% set starttime now/d
[basecm11->...ck*[Jobs completed this day grouped by user*]]% set endtime now/d
[basecm11->...ck*[Jobs completed this day grouped by user*]]% commit
```

Chargeback Groupings

The grouping for the new chargeback `Jobs completed this day grouped by user` is set to be `Group by user`. Grouping by user is a common grouping, because finding resource use by an individual user is typically the most useful case. Grouping is possible by:

- user
- group
- account
- job name
- job ID
- accounting info

After setting up chargebacks to suit the needs of the cluster administrator, queries can be made and reports can be generated using chargebacks. The `report` and `request` commands (section 13.2.3) are used for this.

13.2.3 The `report` And `request` Commands

Continuing on with the chargeback `jobs completed this day grouped by user` created in section 13.2.2, the `report` and `request` commands can be used after CPU request data values have been gathered on the jobs being run.

The `report` Command And Its Options

The `report` command displays a table of the number of jobs that were run per grouping for a chargeback, alongside the resource use and cost for each grouping.

Thus, some time after running jobs in Slurm, the `report` output for the chargeback created earlier, `jobs completed this day grouped by user`, might look as follows (some columns elided for clarity):

Example

```
[basecm11->wlm[slurm]->chargeback[jobs completed this day grouped by user]]% report
# Start - Tue Sep  8 00:00:00 2020 CEST (1599516000)
# End   - Tue Sep  8 23:59:59 2020 CEST (1599602399)
User         Jobs         Runtime (s)  CPU (s)      CPU ($)    ...  Price ($)
------------ ------------ ------------ ------------ ---------- ...  ----------
alice        17           7,806        7,806        0.78       ...  0.78
bob          20           8,775        8,775        0.88       ...  0.88
charlie      19           7,007        7,007        0.7        ...  0.7
david        10           5,122        5,122        0.51       ...  0.51
edgar        25           10,502       10,502       1.05       ...  1.05
frank        21           8,289        8,289        0.83       ...  0.83
```

The help text for `report` command lists formatting options:

```
[basecm11->wlm[slurm]->chargeback]% help report
Name:
          report - Create charge back report
```

```
Usage:
          report [options] <name>

Options:
          -d, --delimiter
               Set default row separator

          -v, --verbose
               Be more verbose: multiline table

          --start
               Pagination start offset

          --limit
               Pagination result limit
```

The `request` Command And Its Options

The `request` command lists the chargeback resources requested for a workload manager.

The `request` command can be run without options. In that case the output shows the resource request consumption for the chargeback, for the jobs over the period associated with that chargeback:

Example

```
[basecm11->wlm[slurm]->chargeback[jobs completed this day grouped by user]]% request
# Start - Sun Sep 13 00:00:00 2020 CEST (1599948000)
# End   - Sun Sep 13 23:59:59 2020 CEST (1600034399)
Jobs         Runtime (s)  CPU (s)      Core (s)     GPU (s)      Slots (s)    Memory (B*s)
------------ ------------ ------------ ------------ ------------ ------------ ------------
5            1,765        1,765        0            0            0            0
```

The help text for `request` lists formatting, grouping, pricing, and filtering options:

```
[basecm11->wlm[slurm]->chargeback]% help request
Name:
          request - Chargeback report for workload

Usage:

          request [options]

Options:
          -u, --group-by-user
               group by username

          --filter-user <user>[,<user>,...]
               filter on specified users

          -g, --group-by-group
               group by group name

          --filter-group <group>[,<group>,...]
               filter on specified groups

          -a, --group-by-account
               group by account
```

```
--filter-account <account>[,<account>,...]
     filter on specified accounts

-j, --group-by-job-name
     group by job name

--filter-job-name <job-name>[,<job-name>,...]
     filter on specified job names

-i, --group-by-job-id
     group by job id

--filter-job-id <job-id>[,<job-id>,...]
     filter on specified job ids

--group-by-accounting-info <field>[,<field>,...]
     group by specified accounting info fields

--filter-accounting-info <field=value>[,<field=value>,...]
     filter on specified parent-ids

--price-per-cpu-second
     Price per CPU second

--price-per-cpu-core-second
     Price per CPU core second

--price-per-gpu-second
     Price per GPU second

--price-per-gpu-second
     Price per GPU second

--price-per-memory-byte-second
     Price per memory byte * second

--price-per-slot-second
     Price per slot second

--currency
     Change the currency in which the price is displayed (default $)

--include-running
     include running jobs in charge back report (prices will not be final)

--calculate-prediction
     calculate a prediction for an incomplete time frame

-d, --delimiter
     Set default row separator

--sort <field1>[,<field2>,...]
     Override default sort order
```

```
          --start-time, -s <time>
               Start time in Prometheus format

          --end-time, -e <time>
               End time in Prometheus format, falls back to start-time if not specified

          --utc
               Use UTC instead of local time

          --start
               Pagination start offset

          --limit
               Pagination result limit

          -v, --verbose
               Be more verbose

Examples:
          request             Chargeback report for workload for the current WLM
          request default     Chargeback report for workload for default
```

For example, using epoch times to specify the start and end times, and grouping by user:

Example

```
[basecm11->wlm[slurm]->chargeback[jobs completed this day grouped by user]]% request -s \
1599948000 -e 1600034399 -u
# Start - Sun Sep 13 00:00:00 2020 CEST (1599948000)
# End   - Sun Sep 13 23:59:59 2020 CEST (1600034399)
User      Jobs  Runtime (s)  CPU (s)      Core (s)  GPU (s)    Slots (s)    Memory (B*s)
--------- ----- ------------ ------------ --------- ---------- ------------ ------------
alice     1     398          398          0         0          0            0
david     2     1,041        1,041        0         0          0            0
edgar     1     325          325          0         0          0            0
frank     1     1            1            0         0          0            0
```

Another way of duplicating the output of `request` without options, is to explicitly specify the default values. For the chargeback `jobs completed this day grouped by user`, which was set up in section 13.2.2. it corresponds to a start time of `now/d` and an end time of `now/d`:

Example

```
[basecm11->...eback[jobs completed this day grouped by user]]% request
# Start - Tue Sep 15 00:00:00 2020 CEST (1600120800)
# End   - Tue Sep 15 23:59:59 2020 CEST (1600207199)
Jobs         Runtime (s)  CPU (s)      Core (s)     GPU (s)      Slots (s)    Memory (B*s)
------------ ------------ ------------ ------------ ------------ ------------ ------------
56           72,374       72,374       0            0            0            0
[basecm11->...eback[jobs completed this day grouped by user]]% request -s now/d -e now/d
# Start - Tue Sep 15 00:00:00 2020 CEST (1600120800)
# End   - Tue Sep 15 23:59:59 2020 CEST (1600207199)
Jobs         Runtime (s)  CPU (s)      Core (s)     GPU (s)      Slots (s)    Memory (B*s)
------------ ------------ ------------ ------------ ------------ ------------ ------------
56           72,374       72,374       0            0            0            0
```

Users can be added to the table with the -u option (some output is truncated here for clarity in the examples that follow):

Example

```
[basecm11->...eback[jobs completed this day grouped by user]]% request -u
# Start - Tue Sep 15 00:00:00 2020 CEST (1600120800)
# End   - Tue Sep 15 23:59:59 2020 CEST (1600207199)
User         Jobs         Runtime (s)  CPU (s)      Core (s)     GPU (s)
------------ ------------ ------------ ------------ ------------ ------------
alice        16           19,538       19,538       0            0
bob          7            15,158       15,158       0            0
charlie      6            6,056        6,056        0            0
david        3            1,636        1,636        0            0
edgar        15           16,475       16,475       0            0
frank        9            13,511       13,511       0            0
```

A jobs drilldown can be carried out with -j.

Example

```
[basecm11->wlm[slurm]->chargeback[jobs completed this day grouped by user]]% request  -j
# Start - Tue Sep 15 00:00:00 2020 CEST (1600120800)
# End   - Tue Sep 15 23:59:59 2020 CEST (1600207199)
Job name       Jobs         Runtime (s)  CPU (s)      Core (s)     GPU (s)
-------------- ------------ ------------ ------------ ------------ ------------
data-transfer  15           49,248       49,248       0            0
iozone         19           9,918        9,918        0            0
sleep          22           13,208       13,208       0            0
```

The jobs drilldown can be carried out for a particular user, alice, using the filter option:

Example

```
[basecm11->...grouped by user]]% request -j --filter-user alice
# Start - Tue Sep 15 00:00:00 2020 CEST (1600120800)
# End   - Tue Sep 15 23:59:59 2020 CEST (1600207199)
Job name       Jobs         Runtime (s)  CPU (s)      Core (s)     GPU (s)
-------------- ------------ ------------ ------------ ------------ ------------
data-transfer  5            13,404       13,404       0            0
iozone         6            3,132        3,132        0            0
sleep          5            3,002        3,002        0            0
```

This can have the user fields added for the case when several users are specified, as follows:

Example

```
[basecm11->...grouped by user]]% request -s now/d -e now/d -u -j --filter-user alice,bob,charlie
# Start - Tue Sep 15 00:00:00 2020 CEST (1600120800)
# End   - Tue Sep 15 23:59:59 2020 CEST (1600207199)
User         Job name       Jobs         Runtime (s)  CPU (s)      Core (s)     GPU (s)
------------ -------------- ------------ ------------ ------------ ------------ ------------
alice        data-transfer  5            13,404       13,404       0            0
alice        iozone         6            3,132        3,132        0            0
alice        sleep          5            3,002        3,002        0            0
bob          data-transfer  3            12,842       12,842       0            0
bob          iozone         1            516          516          0            0
bob          sleep          3            1,800        1,800        0            0
```

```
charlie      data-transfer  1            3,212        3,212        0            0
charlie      iozone         2            1,044        1,044        0            0
charlie      sleep          3            1,800        1,800        0            0
[basecm11->wlm[slurm]->chargeback[jobs completed this day grouped by user]]%
```

As the help text for `request` suggests, there are many more combinations possible.

13.3 Job Chargeback Background Information

Because users can run large numbers of jobs per day, the storage requirement for chargeback records can be very large indeed. For this reason, a cache is kept in memory, and flushed to storage in a MySQL database.

CMDaemon `AdvancedConfig` directives for configuring the MySQL storage for chargebacks are:

- `JobInformationChargeBackKeepDuration` (page 935)
- `JobInformationChargeBackKeepCount` (page 935)
- `JobInformationChargeBackRemoveInterval` (page 936)

14

Day-to-day Administration

Just as for regular Linux system administration, it is a best practice for cluster administration procedures to be documented as they are carried out.

Updating software packages for bug fixes or for security fixes is a substantial and important part of the tasks carried out in daily cluster administration. It has its own chapter (Chapter 9).

This chapter discusses other tasks that may come up in day-to-day cluster administration with NVIDIA Base Command Manager.

Section 14.1 discusses running shell commands in parallel over the cluster.

Section 14.2 discusses how a cluster administrator can ask the help of the BCM support team for guidance with an issue in an optimum manner.

Section 14.3 discusses how backups can be implemented for BCM.

Section 14.4 discusses revision control for images.

Section 14.5 discusses BIOS configuration with BCM.

Section 14.6 discusses checking hardware matching across the nodes of the cluster.

Section 14.7 discusses Serial Over LAN console access.

Section 14.8 discusses administrative aspects of handling the large amounts of raw monitoring data.

Section 14.9 discusses node replacement.

Section 14.10 discusses using Ansible to configure the cluster via Ansible collections and playbooks.

14.1 Parallel Shells: `pdsh` And `pexec`

What `pdsh` And `pexec` Do

The cluster management tools include two parallel shell execution commands:

- `pdsh` (parallel distributed shell, section 14.1.1), runs from within the OS shell. That is, `pdsh` executes its commands from within `bash` by default.
- `pexec` (parallel execute, section 14.1.2, runs from within CMDaemon. That is, `pexec` executes its commands from within the `cmsh` front end.

A one-time execution of `pdsh` or `pexec` can run one or more shell commands on a group of nodes in parallel.

A Warning About Power Surge Risks With `pdsh` And `pexec`

Some care is needed when running `pdsh` or `pexec` with commands that affect power consumption. For example, running commands that power-cycle devices in parallel over a large number of nodes can be risky because it can put unacceptable surge demands on the power supplies.

Within `cmsh`, executing a `power reset` command from `device` mode to power cycle a large group of nodes is much safer than running a parallel command to do a reset using `pdsh` or `pexec`. This is because the CMDaemon power reset powers up nodes after a deliberate delay between nodes (section 4.2).

Which Command To Use Out Of pdsh And pexec

The choice of using pdsh or pexec commands is mostly up to the administrator. The only time that running pdsh from bash is currently required instead of running pexec from within cmsh, is when stopping and restarting CMDaemon on a large number of regular nodes (section 2.6.1). This is because a command to stop CMDaemon on a regular node, that itself relies on being run from a running CMDaemon on a regular node, can obviously give unexpected results.

14.1.1 pdsh In The OS Shell

Packages Required For pdsh

By default, the following packages must be installed from the BCM repositories to the head node for pdsh to function fully:

- pdsh
- genders
- pdsh-mod-cmupdown
- pdsh-mod-genders
- pdsh-rcmd-exec
- pdsh-ssh

The pdsh utility is modular, that is, it gets extra functionality through the addition of modules.

The genders package is used to generate a default /etc/genders configuration file. The file is used to decide what and how nodes are to be used or excluded by default, when used with pdsh. Configuration details can be found in man pdsh(1). The configuration can probably best be understood by viewing the file itself and noting that BCM in the default configuration associates the following *genders* with a list of nodes:

- all: all the nodes in the cluster, head and regular nodes.
- category=default: the nodes that are in the default category
- computenode: regular nodes
- headnode: node or nodes that are head nodes.

In a newly-installed cluster using default settings, the genders category=default and computenode have the same list of nodes to begin with.

The default /etc/genders file has a section that is generated and maintained by CMDaemon, but the file can be altered by the administrator outside the CMDaemon-maintained section. However, it is not recommended to change the file manually frequently. For example, tracking node states with this file is not recommended. Instead, the package pdsh-mod-cmupdown provides the -v option for node state tracking functionality, and how to use this and other pdsh options is described in the next section.

pdsh Options

In the OS shell, running pdsh -h displays the following help text:

```
Usage: pdsh [-options] command ...
-S                return largest of remote command return values
-h                output usage menu and quit
-V                output version information and quit
-q                list the option settings and quit
-b                disable ^C status feature (batch mode)
-d                enable extra debug information from ^C status
```

```
-l user           execute remote commands as user
-t seconds        set connect timeout (default is 10 sec)
-u seconds        set command timeout (no default)
-f n              use fanout of n nodes
-w host,host,...  set target node list on command line
-x host,host,...  set node exclusion list on command line
-R name           set rcmd module to name
-M name,...       select one or more misc modules to initialize first
-N                disable hostname: labels on output lines
-L                list info on all loaded modules and exit
-v                exclude targets if they are down
-g query,...      target nodes using genders query
-X query,...      exclude nodes using genders query
-F file           use alternate genders file `file'
-i                request alternate or canonical hostnames if applicable
-a                target all nodes except those with "pdsh_all_skip" attribute
-A                target all nodes listed in genders database
available rcmd modules: ssh,exec (default: ssh)
```

Further options and details are given in `man pdsh(1)`.

Examples Of `pdsh` Use

For the examples in this section, a cluster can be considered that is set up with two nodes, with the state of node001 being `UP` and that of node002 being `DOWN`:

```
[root@basecm11 ~]# cmsh -c "device status"
node001 .................. [   UP   ]
node002 .................. [ DOWN   ]
basecm11 ................. [   UP   ]
```

In the examples, the outputs for `pdsh` could be as follows for the `pdsh` options considered:

-A: With this `pdsh` option an attempt is made to run the command on all nodes, regardless of the node state:

Example

```
[root@basecm11 ~]# pdsh -A hostname
node001: node001
node002: ssh: connect to host node002 port 22: No route to host
pdsh@basecm11: node002: ssh exited with exit code 255
basecm11: basecm11
```

-v: With this option an attempt is made to run the command only on nodes nodes that are in the state `UP`:

Example

```
[root@basecm11 ~]# pdsh -A -v hostname
node001: node001
basecm11: basecm11
```

-g: With this option, and using, for example, `computenode` as the genders query, only nodes within `computenode` in the `/etc/genders` file are considered for the command. The `-v` option then further ensures that the command runs only on a node in `computenode` that is up. In a newly-installed cluster, regular nodes are assigned to `computenode` by default, so the command runs on all regular nodes that are up in a newly-installed cluster:

Example

```
[root@basecm11 ~]# pdsh -v -g computenode hostname
node001: node001
```

-w: This option allows a node list (`man pdsh(1)`) to be specified on the command line itself:

Example

```
[root@basecm11 ~]# pdsh -w node00[1-2] hostname
node001: node001
node002: ssh: connect to host node002 port 22: No route to host
pdsh@basecm11: node002: ssh exited with exit code 255
```

-x: This option is the converse of `-w`, and excludes a node list that is specified on the command line itself:

Example

```
[root@basecm11 ~]# pdsh -x node002 -w node00[1-2] hostname
node001: node001
```

The `dshbak` Command

The `dshbak` (distributed shell backend formatting filter) command is a filter that reformats `pdsh` output. It comes with the `pdsh` package.

Running `dshbak` with the `-h` option displays:

```
[root@basecm11 ~]# dshbak -h
Usage: dshbak [OPTION]...
 -h       Display this help message
 -c       Coalesce identical output from hosts
 -d DIR   Send output to files in DIR, one file per host
 -f       With -d, force creation of DIR
```

Further details can be found in `man dshbak(1)`.

For the examples in this section, it is assumed that all the nodes in the cluster are now up. That is, `node002` used in the examples of the preceding section is now also up. Some examples to illustrate how `dshbak` works are then the following:

Without `dshbak`:

Example

```
[root@basecm11 ~]# pdsh -A ls /etc/services /etc/yp.conf
basecm11: /etc/services
basecm11: /etc/yp.conf
node001: /etc/services
node001: ls: cannot access /etc/yp.conf: No such file or directory
pdsh@basecm11: node001: ssh exited with exit code 2
node002: /etc/services
node002: /etc/yp.conf
```

With dshbak, with no dshbak options:

Example

```
[root@basecm11 ~]# pdsh -A ls /etc/services /etc/yp.conf | dshbak
node001: ls: cannot access /etc/yp.conf: No such file or directory
pdsh@basecm11: node001: ssh exited with exit code 2
----------------
basecm11
----------------
/etc/services
/etc/yp.conf
----------------
node001
----------------
/etc/services
----------------
node002
----------------
/etc/services
/etc/yp.conf
[root@basecm11 ~]#
```

With dshbak, with the -c (coalesce) option:

Example

```
[root@basecm11 ~]# pdsh -A ls /etc/services /etc/yp.conf | dshbak -c
node001: ls: cannot access /etc/yp.conf: No such file or directory
pdsh@basecm11: node001: ssh exited with exit code 2
----------------
node002,basecm11
----------------
/etc/services
/etc/yp.conf
----------------
node001
----------------
/etc/services
[root@basecm11 ~]#
```

The dshbak utility is useful for creating human-friendly output in clusters with larger numbers of nodes.

14.1.2 pexec In cmsh

In cmsh, the pexec command is run from device mode:

Example

```
[basecm11->device]% pexec -n node001,node002 "cd ; ls"

[node001] :
anaconda-ks.cfg
install.log
install.log.syslog
```

```
[node002] :
anaconda-ks.cfg
install.log
install.log.syslog
```

14.1.3 pexec In Base View

In Base View, pexec is hidden, but executed in a GUI wrapper, using the navigation path Cluster > Run command.

For large numbers of nodes, rendering the output into the node subpanes (little boxes) can take a long time. To improve the Base View experience, selecting the Single text view icon instead of the Grouped view icon speeds up the rendering significantly, but at the cost of removing the borders of the subpanes.

Ticking the Join output checkbox places output that is the same for particular nodes, into the same subpane.

Running parallel shell commands from cmsh instead of in Base View is faster in most cases, due to less graphics rendering overhead.

14.1.4 Using The -j|--join Option Of pexec In cmsh

The output of the pexec command by default can come out in a sequence depending on node response time. To make it more useful for an administrator, order can be imposed on the output. Checking consistency across nodes is then easier.

For example, in a cluster with 2 nodes, the /etc/resolv.conf files for each node could be displayed as:

Example

```
[basecm11->device]% pexec -c default "cat /etc/resolv.conf"

[node001] :
# This file was generated by the Node Installer.
search cm.cluster eth.cluster brightcomputing.com
nameserver 10.141.255.254

[node002] :
# This file was generated by the Node Installer.
search cm.cluster eth.cluster brightcomputing.com
nameserver 10.141.255.254
```

More order can be imposed on the preceding output by using the -j|--join option. This joins identical fields together in a way similar to the standard unix text utility, join, which makes the result easier to view:

Example

```
[basecm11->device]% pexec -j -c default "cat /etc/resolv.conf"
[node001,node002]
# This file was generated by the Node Installer.
search cm.cluster eth.cluster brightcomputing.com
nameserver 10.141.255.254
```

In the following example, a cluster with 10 nodes is inspected. In the cluster, node002 is down, and the idea is to see if the remaining nodes have the same mounts:

Example

```
[basecm11->device]% pexec -j -c default "mount|sort"
Nodes down:         node002
[node002]
Node down

[node001,node003..node010]
/dev/hda1 on / type ext3 (rw,noatime,nodiratime)
/dev/hda2 on /var type ext3 (rw,noatime,nodiratime)
/dev/hda3 on /tmp type ext3 (rw,nosuid,nodev,noatime,nodiratime)
/dev/hda6 on /local type ext3 (rw,noatime,nodiratime)
master:/cm/shared on /cm/shared type nfs
(rw,rsize=32768,wsize=32768,hard,intr,addr=10.141.255.254)
master:/home on /home type nfs
(rw,rsize=32768,wsize=32768,hard,intr,addr=10.141.255.254)
none on /dev/pts type devpts (rw,gid=5,mode=620)
none on /dev/shm type tmpfs (rw)
none on /proc/sys/fs/binfmt_misc type binfmt_misc (rw)
none on /proc type proc (rw,nosuid)
none on /sys type sysfs (rw)
```

Here, even more order is imposed by sorting the output of each mount command within `bash` before the `-j` option operates from `cmsh`. The `-c` option executes the command on the `default` category of nodes.

14.1.5 Other Parallel Commands

Besides `pexec`, CMDaemon has several other parallel commands:

pkill: parallel kill
Synopsis:
`pkill [OPTIONS] <tracker> [<tracker> ... ]`

plist: List the parallel commands that are currently running, with their tracker ID
Synopsis:
`plist`

pping: ping nodes in parallel
Synopsis:
`pping [OPTIONS]`

pwait: wait for parallel commands that are running to complete
Synopsis:
`pwait [OPTIONS] <tracker> [<tracker> ...]`

Details on these parallel commands, including examples, can be seen by executing the `help` command within the `device` mode of `cmsh` for a parallel command, <*pcommand*>, as follows:

```
[basecm11->device]%help <pcommand>
```

14.2 Getting Support With BCM Issues, And Notifications For Release Updates

The scope of BCM technical support is described in Appendix C of the *Installation Manual*.

14.2.1 The Support Portal For BCM

Support requests can be sent in via the support portal at
`https://enterprise-support.nvidia.com/s/create-case`

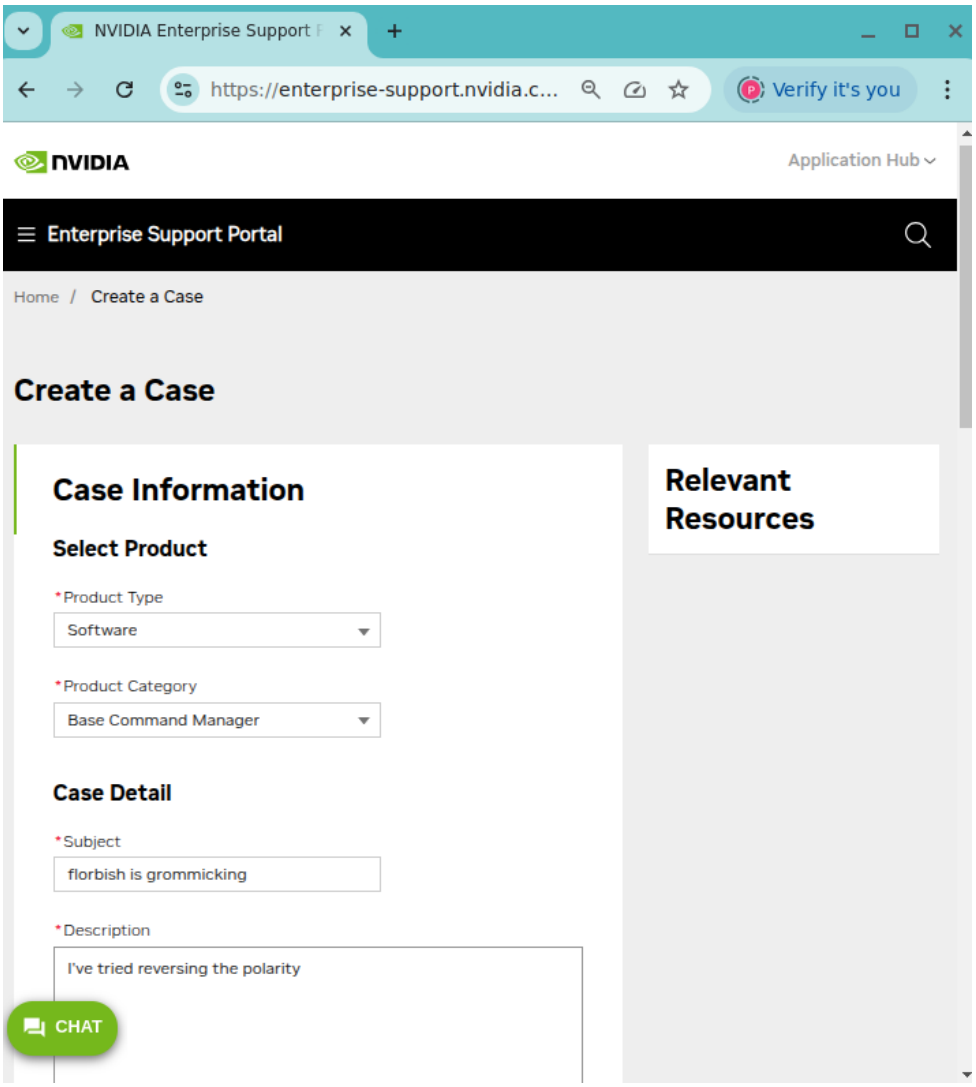


Figure 14.1: Customer support portal: submitting a support request

When creating a BCM support case at that URL:

- the `Product Type` that must be selected is `Software`.
- the `Product Category` that must be selected is:
  - `Bright Cluster Manager` for issues related to versions of BCM prior to BCM version 10.
  - `Base Command Manager` for issues related to BCM version 10 and beyond.
- Registered users are advised to log in to the support portal. It makes the user experience better because cluster-related data values are already filled in for a logged-in user.
- Unregistered users can also create a case via the same URL, to deal with registration issues.

The Enterprise Support Portal User Guide at

https://enterprise-support.nvidia.com/s/article/NVIDIA-Enterprise-Support-Guide-for-New-Users

has further details on how users can submit support requests.

BCM support as implemented via the support portal is carried out primarily via e-mail. To help resolve issues, the `cm-diagnose` (section 14.2.2) utility is provided as a supplement, and is discussed next.

14.2.2 Reporting Cluster Manager Diagnostics With `cm-diagnose`

The diagnostic utility `cm-diagnose` is run from the head node. It gathers data on the cluster that may help diagnose issues. Running it as

`cm-diagnose --help`

displays its options, capabilities, and defaults.

For particular issues it may be helpful to change some of the default values to gather data in a more targeted way. For example, for larger clusters, the log file limit and the timeout values can both be increased to avoid logs being cropped and commands being killed during the gathering of diagnostic data:

Example

```
[root@basecm11 ~]# cm-diagnose --limit 100 --timeout 240
```

The preceding command increases the log limit from its default of 50MB to 100MB, and the timeout value from its default of 60s to 240s.

When carrying out a run to pick up diagnostic data, `cm-diagnose` runs interactively by default. The administrator can send the resultant diagnostics file to BCM support directly. The output of a `cm-diagnose` session looks something like the following (the output has been made less verbose for easy viewing):

Example

```
[root@basecm11 ~]# cm-diagnose
To be able to contact you about the issue, please provide
your e-mail address (or support ticket number, if applicable):
franknfurter@example.com

Please enter the related Support Request (SR) or NVIDIA Enterprise Support number and any other
related additional information: [SR-XXXXX] or [XXXXXXXX]
End input with ctrl-d
This has to do with SR-999999 that I submitted just now. In short:
I tried X, Y, and Z on the S2464 motherboard. When that didn't work, I
tried A, B, and C, but the florbish is grommicking.
Thank you.

Support Request number: SR-999999
Thank you.
If issues are suspected in the cmdaemon process, a gdb trace of that process
is useful. In general such trace is only needed if Bright Support asks for this.
Do you want to create a gdb trace of the running CMDaemon? [y/N]

Proceed to collect information? [Y/n]

Processing master
```

```
    Processing commands
        /bin/uname -a
        /usr/bin/top -b -n 1
        /sbin/ifconfig -a
        ...
    Processing file contents
        ...
    Processing large files and log files
        ...
    Collecting process information for CMDaemon
        gdb -p 1334
    Executing CMSH commands
        ...
    Finished executing CMSH commands

Processing default-image
    Processing commands
        ...
    Processing file contents
        ...
Creating log file: /root/SR-999999-basecm11__1234.tar.gz

Cleaning up

Automatically submit diagnostics file to http://support.brightcomputing.com/cm-diagnose/ ? [Y/n]

Uploaded file: SR-999999-basecm11__1234.tar.gz
Remove log file (/root/SR-999999-basecm11__1234.tar.gz)?  [y/N] y
[root@basecm11 ~]#
```

14.2.3 Getting Notified About Updates

Updates for the various NVIDIA Base Command Manager releases continue for some time after the initial release is made public. The updates typically have bugfixes and improvements, and an administrator may wish to install an update when it becomes publicly available.

The release notes for BCM can be found online at:

`https://docs.nvidia.com/base-command-manager/#release-notes`

14.3 Backups

14.3.1 Cluster Installation Backup

BCM does not include facilities to create backups of a cluster installation. The cluster administrator is responsible for deciding on the best way to back up the cluster, out of the many possible choices.

A backup method is strongly recommended, and checking that restoration from backup actually works is also strongly recommended.

One option that may be appropriate for some cases is simply cloning the head node. A clone can be created by PXE booting the new head node, and following the procedure in section 15.4.8.

When setting up a backup mechanism, it is recommended that the full filesystem of the head node (i.e. including all software images) is backed up. Unless the regular node hard drives are used to store important data, it is not necessary to back them up.

If no backup infrastructure is already in place at the cluster site, the following open source (GPL) software packages may be used to maintain regular backups:

- **Bacula**: Bacula is a mature network based backup program that can be used to backup to a remote storage location. If desired, it is also possible to use Bacula on nodes to back up relevant data that

is stored on the local hard drives. More information is available at http://www.bacula.org

Bacula requires ports 9101-9103 to be accessible on the head node. The ports can be opened from an IP address of 93.184.216.34 on the external network using the openports mode of the firewall role in cmsh:

Example

```
[basecm11->device[basecm11]->roles[firewall]->openports]% add ACCEPT net 9101 tcp fw 93.184.216.34/32
[basecm11->device[basecm11]->roles[firewall]->openports]% add ACCEPT net 9102 tcp fw 93.184.216.34/32
[basecm11->device[basecm11]->roles[firewall]->openports]% add ACCEPT net 9103 tcp fw 93.184.216.34/32
[basecm11->device[basecm11]->roles[firewall]->openports]% commit
```

The shorewall service is restarted automatically to enforce the added rules.

- **rsnapshot**: rsnapshot allows periodic incremental filesystem snapshots to be written to a local or remote filesystem. Despite its simplicity, it can be a very effective tool to maintain frequent backups of a system. More information is available at http://www.rsnapshot.org.

  Rsnapshot requires access to port 22 on the head node.

14.3.2 Local Database And Data Backups And Restoration

The CMDaemon database is stored in the MySQL cmdaemon database, and contains most of the stored settings of the cluster.

Monitoring data values are stored as binaries in the filesystem, under /var/spool/cmd/monitoring.

The administrator is expected to run a regular backup mechanism for the cluster to allow restores of all files from a recent snapshot. As an additional, separate, convenience:

- For the CMDaemon database:
  - the entire database is, by default, also backed up nightly on the cluster filesystem itself ("local rotating backup") for the last 7 days:

    Example

    ```
    [root@basecm11 ~]# ls -1t /var/spool/cmd/backup/
    backup-Wed.sql.gz
    backup-Tue.sql.gz
    backup-Mon.sql.gz
    backup-Sun.sql.gz
    backup-Sat.sql.gz
    backup-Fri.sql.gz
    backup-Thu.sql.gz
    ```

  - the CMDaemon database can also be manually backed up with the cmdaemon-backup command. This can be useful if the administrator would like to carry out an extensive change on the cluster and would like the reassuring possibility of getting back the old configuration. If no argument is supplied to cmdaemon-backup, then any existing default backup of the day is overwritten. If a string is supplied, then the default backup- prefix is replaced. The replacement becomes the supplied string concatenated with a timestamp. This is most easily illustrated with an example:

    Example

    ```
    [root@basecm11 ~]# cmdaemon-backup manual
    [root@basecm11 ~]# ls /var/spool/cmd/backup/manual*
    manual-24-08-28_13-19-59_Wed.sql.gz
    ```

The time stamp in the preceding example is of the form:
-<two-digit year>-<month>-<date>_<hour>-<minute>-<seconds>_

- For the monitoring data, the raw data records are not backed up locally, since these can get very large. However, the configuration of the monitoring data, which is stored in the CMDaemon database, is backed up for the last 7 days too.

Database Corruption Messages

A corrupted MySQL database is commonly caused by an improper shutdown of the node. To deal with this, when starting up, MySQL checks itself for corrupted tables, and tries to repair itself.

- If MySQL cannot start, then CMDaemon on the head node cannot start.
- If MySQL can start, but corruption is detected in the database later on, then it keeps running in read-only mode. The `mysql` health check on the head node fails, and an `info` message (section 10.10.4) with an indication of the issue is seen. More details can be found in `/var/log/cmdaemon` and using `service status mysql.service` or `journalctl -xeu mysql.service`.

  The `journalctl` output might show something similar to the following output extract, if corruption is detected in the data (some output ellipsized):

  Example

```
[root@basecm11 ~]# journalctl -xeu mysql.service
...
mysqld[179281]:..546 [ERROR] [MY-012224] [InnoDB] Checksum mismatch in datafile: ./cmdaemon/....
mysqld[179281]:..546 [ERROR] [MY-012592] [InnoDB] Operating system error number 22 in a file....
mysqld[179281]:..546 [ERROR] [MY-012596] [InnoDB] Error number 22 means 'Invalid argument'  ...
mysqld[179281]:..546 [ERROR] [MY-012131] [InnoDB] Could not find a valid tablespace file for....
mysqld[179281]:..546 [Warning] [MY-012049] [InnoDB] Cannot calculate statistics for table `c....
mysqld[179281]:..1192 [Warning] [MY-012049] [InnoDB] Cannot calculate statistics for table `....
mysqld[179281]:..1838 [ERROR] [MY-012224] [InnoDB] Checksum mismatch in datafile: ./cmdaemon....
mysqld[179281]:..1838 [ERROR] [MY-012592] [InnoDB] Operating system error number 22 in a fil....
mysqld[179281]:..1838 [ERROR] [MY-012596] [InnoDB] Error number 22 means 'Invalid argument' ...
mysqld[179281]:..1838 [ERROR] [MY-012131] [InnoDB] Could not find a valid tablespace file fo....
mysqld[179281]:..1838 [Warning] [MY-012049] [InnoDB] Cannot calculate statistics for table `....
mysqld[179281]:..2485 [Warning] [MY-012049] [InnoDB] Cannot calculate statistics for table `....
mysqld[179281]:..3131 [Warning] [MY-012049] [InnoDB] Cannot calculate statistics for table `....
mysqld[179281]:..3778 [ERROR] [MY-012224] [InnoDB] Checksum mismatch in datafile: ./cmdaemon....
mysqld[179281]:..3778 [ERROR] [MY-012592] [InnoDB] Operating system error number 22 in a fil....
```

A corrupt database can continue to run for a while, as write attempts pile up in cache. However, eventually the database crashes and then BCM and other systems that rely on the database cannot carry on. Preventing a database crash is therefore important, which is why automatic checks and repairs are carried out when a database starts up.

To fix a corrupted database, a restoration from backup can be carried out, as explained in the next section.

Restoring From The Local Backup

If the MySQL InnoDB repair tools do not automatically fix the problem, then the issue can normally be resolved with some manual intervention for a failover or non-failover configuration.

If the head node is a part of a failover configuration, the `dbreclone` option (section 15.4.2) should normally provide a CMDaemon and Slurm database that is current. The `dbreclone` option does not clone the monitoring data.

Cloning extra databases: The file `/cm/local/apps/cluster-tools/ha/conf/extradbclone.xml.template` can be used as a template to create a file `extradbclone.xml` in the same directory. The `extradbclone.xml` file can then be used to define an additional database to be cloned.

The `/etc/my.cnf` file can then be updated with a new database by adding the lines:

```
replicate-do-db=<name of database>
binlog-do-db=<name of database>
```

The database can then be cloned with this new MySQL configuration by running

```
cmha dbreclone <passive>
```

where <*passive*> is the hostname of the passive head node.

If the head node is not part of a failover configuration, then a restoration from local backup can be done. The local backup directory is `/var/spool/cmd/backup`, with contents that look like (some text elided):

Example

```
[root@solaris ~]# cd /var/spool/cmd/backup/
[root@solaris backup]# ls -l
total 280
...
-rw------- 1 root root 33804 Oct 10 04:02 backup-Mon.sql.gz
-rw------- 1 root root 33805 Oct  9 04:02 backup-Sun.sql.gz
-rw------- 1 root root 33805 Oct 11 04:02 backup-Tue.sql.gz
...
```

The CMDaemon database snapshots are stored as `backup-<day of week>.sql.gz` In the example, the latest backup available in the listing for CMDaemon turns out to be `backup-Tue.sql.gz`

The latest backup can then be ungzipped and piped into the MySQL database for the user `cmdaemon`. The password, <*password*>, can be retrieved from `/cm/local/apps/cmd/etc/cmd.conf`, where it is configured in the `DBPass` directive (Appendix C).

Example

```
gunzip backup-Tue.sql.gz
systemctl stop cmd #(just to make sure)
mysql -ucmdaemon -p<password> cmdaemon < backup-Tue.sql
```

Running "`systemctl start cmd`" should have CMDaemon running again, this time with a restored database from the time the snapshot was taken. That means, that any changes that were done to BCM after the time the snapshot was taken are no longer implemented.

Monitoring data values are not kept in a database, but in files (section 14.8).

14.4 Revision Control For Images

BCM version 7 introduced support for the implementations of Btrfs provided by the distributions. Btrfs makes it possible to carry out revision control for images efficiently.

14.4.1 Btrfs: The Concept And Why It Works Well In Revision Control For Images

Btrfs, often pronounced "butter FS", is a Linux implementation of a copy-on-write (COW) filesystem.

A COW design for a filesystem follows the principle that, when blocks of old data are to be modified, then the new data blocks are written in a new location (the COW action), leaving the old, now

superseded, copy of the data blocks still in place. Metadata is written to keep track of the event so that, for example, the new data blocks can be used seamlessly with the contiguous old data blocks that have not been superseded.

This is in contrast to the simple overwriting of old data that a non-COW filesystem such as Ext3fs carries out.

A result of the COW design means that the old data can still be accessed with the right tools, and that rollback and modification become a natural possible feature.

"Cheap" *revision control* is thus possible.

Revision control for the filesystem is the idea that changes in the file system are tracked and can be rolled back as needed. "Cheap" here means that COW makes tracking changes convenient, take up very little space, and quick. For an administrator of BCM, cheap revision control is interesting for the purpose of managing software images.

This is because for a non-COW filesystem such as Ext3fs, image variations take a large amount of space, even if the variation in the filesystem inside the image is very little. On the other hand, image variations in a COW filesystem such as Btrfs take up near-minimum space.

Thus, for example, the technique of using initialize and finalize scripts to generate such image variations on the fly (section 3.20.4) in order to save space, can be avoided by using a Btrfs partition to save the full image variations instead.

"Expensive" revision control on non-COW filesystems is also possible. It is merely not recommended, since each disk image takes up completely new blocks, and hence uses up more space. The administrator will then have to consider that the filesystem may become full much earlier. The degree of restraint on revision control caused by this, as well as the extra consumption of resources, means that revision control on non-COW filesystems is best implemented on test clusters only, rather than on production clusters.

14.4.2 Btrfs Availability And Distribution Support

Btrfs has been part of the Linux kernel since kernel 2.6.29-rc1. Depending on which Linux distribution is being used on a cluster, it may or may not be a good idea to use Btrfs in a production environment, as in the worst case it could lead to data loss.

Btrfs has been officially removed from RHEL distributions since RHEL8.

Btrfs features are supported in SLES15, as described at `https://www.suse.com/releasenotes/x86_64/SUSE-SLES/15-SP1/index.html#TechInfo.Filesystems`.

While no problems have been noticed with storing software images on Btrfs using BCM, it is highly advisable to keep backups of important software images on a non-Btrfs filesystem when Btrfs is used.

An issue with using `cm-clone-install` with Btrfs is described on page 828.

14.4.3 Installing Btrfs To Work With Revision Control Of Images In BCM

Installation Of `btrfs-progs`

To install a Btrfs filesystem, the `btrfs-progs` packages must be installed from the distribution repository first (some lines elided):

Example

```
[root@basecm11 ~]# yum install btrfs-progs
...
Resolving Dependencies
--> Running transaction check
---> Package btrfs-progs.x86_64 0:4.9.1-1.el7 will be installed
...
Total download size: 678 k
Installed size: 4.0 M
...
Complete!
```

Creating A Btrfs Filesystem

The original images directory can be moved aside first, and a new images directory created to serve as a future mount point for Btrfs:

Example

```
[root@basecm11 ~]# cd /cm/
[root@basecm11 cm]# mv images images2
[root@basecm11 cm]# mkdir images
```

A block device can be formatted as a Btrfs filesystem in the usual way by using the `mkfs.btrfs` command on a partition, and then mounted to the new images directory:

Example

```
[root@basecm11 cm]# mkfs.btrfs /dev/sdc1
[root@basecm11 cm]# mount /dev/sdc1 /cm/images
```

If there is no spare block device, then, alternatively, a file with zeroed data can be created, formatted as a Btrfs filesystem, and mounted as a loop device like this:

Example

```
[root@basecm11 cm]# dd if=/dev/zero of=butter.img bs=1G count=20
20+0 records in
20+0 records out
21474836480 bytes (21 GB) copied, 916.415 s, 23.4 MB/s
[root@basecm11 cm]# mkfs.btrfs butter.img

WARNING! - Btrfs Btrfs v0.20-rc1 IS EXPERIMENTAL
WARNING! - see http://btrfs.wiki.kernel.org before using

fs created label (null) on butter.img
        nodesize 4096 leafsize 4096 sectorsize 4096 size 20.00GB
Btrfs Btrfs v0.20-rc1
[root@basecm11 cm]# mount -t btrfs butter.img images -o loop
[root@basecm11 cm]# mount
...
/cm/butter.img on /cm/images type btrfs (rw,loop=/dev/loop0)
```

Migrating Images With `cm-migrate-images`

The entries inside the `/cm/images/` directory are the software images (file trees) used to provision nodes.

Revision tracking of images in a Btrfs filesystem can be done by making the directory of a specific image a *subvolume*. Subvolumes in Btrfs are an extension of standard unix directories with versioning. The files in a subvolume are tracked with special internal Btrfs markers.

To have this kind of version tracking of images work, image migration cannot simply be done with a `cp -a` or a `mv` command. That is, moving images from the`images2` directory in the traditional filesystem, over to images in the `images` directory in the Btrfs filesystem command with the standard `cp` or `mv` command is not appropriate. This is because the images are to be tracked once they are in the Btrfs system, and are therefore not standard files any more, but instead extended, and made into subvolumes.

The migration for BCM software images can be carried out with the utility `cm-migrate-image`, which has the usage:

```
cm-migrate-image <path to old image> <path to new image>
```

where the old image is a traditional directory, and the new image is a subvolume.

Example

```
[root@basecm11 cm]# cm-migrate-image /cm/images2/default-image /cm/images/default-image
```

The `default-image` directory, or more exactly, subvolume, must not exist in the Btrfs filesystem before the migration. The subvolume is created only for the image basename that is migrated, which is `default-image` in the preceding example.

In the OS, the `btrfs` utility is used to manage Btrfs. Details on what it can do can be seen in the `btrfs(8)` man page. The state of the loop filesystem can be seen, for example, with:

Example

```
[root@basecm11 cm]# btrfs filesystem show /dev/loop0
Label: none  uuid: 6f94de75-2e8d-45d2-886b-f87326b73474
        Total devices 1 FS bytes used 3.42GB
        devid    1 size 20.00GB used 6.04GB path /dev/loop0

Btrfs Btrfs v0.20-rc1
[root@basecm11 cm]# btrfs subvolume list /cm/images
ID 257 gen 482 top level 5 path default-image
```

The filesystem can be modified as well as merely viewed with `btrfs`. However, instead of using the utility to modify image revisions directly, it is recommended that the administrator use BCM to manage image version tracking, since the necessary functionality has been integrated into `cmsh` and Base View.

14.4.4 Using `cmsh` For Revision Control Of Images

Revision Control Of Images Within `softwareimage` Mode

The following commands and extensions can be used for revision control in the `softwareimage` mode of `cmsh`:

- `newrevision` <*parent software image name*> "*textual description*"
  Creates a new revision of a specified software image. For that new revision, a revision number is automatically generated and saved along with time and date. The new revision receives a name of the form:

  <*parent software image name*>`@`<*revision number*>

  A new Btrfs subvolume:

  `/cm/images/`<*parent software image name*>`-`<*revision number*>

  is created automatically for the revision. From now on, this revision is a self-contained software image and can be used as such.

- `revisions [-a|--all]` <*parent software image name*>
  Lists all revisions of specified parent software image in the order they ware created, and associates the revision with the revision number under the header `ID`. The option `-a|--all` also lists revisions that have been removed.

- `list [-r|--revisions]`
  The option `-r|--revisions` has been added to the `list` command. It lists all revisions with their name, path, and kernel version. A parent image is the one at the head of the list of revisions, and does not have `@`<*revision number*> in its name.

- `setparent` [*parent software image name*] <*revision name*>
  Sets a revision as a new parent. The action first saves the image directory of the current, possibly altered, parent directory or subvolume, and then attempts to copy or snapshot the directory or subvolume of the revision into the directory or subvolume of the parent. If the attempt fails, then it tries to revert all the changes in order to get the directory or subvolume of the parent back to the state it was in before the attempt.

- `remove [-a|--all] [-d|--data]` <*parent software image name*>
  Runnning `remove` without options removes the parent software image. The option `-a|--all` removes the parent and all its revisions. The option `-d|--data` removes the actual data. To run the `remove` command, any images being removed should not be in use.

Revision Control Of Images Within `category` Mode

The `category` mode of `cmsh` also supports revision control.

Revision control can function in 3 kinds of ways when set for the `softwareimage` property at category level.

To explain the settings, an example can be prepared as follows: It assumes a newly-installed cluster with Btrfs configured and `/cm/images` migrated as explained in section 14.4.3. The cluster has a default software image `default-image`, and two revisions of the parent image are made from it. These are automatically given the paths `default-image@1` and `default-image@2`. A category called `storage` is then created:

Example

```
[basecm11->softwareimage]% newrevision default-image "some changes"
[basecm11->softwareimage]% newrevision default-image "more changes"
[basecm11->softwareimage]% revisions default-image
ID    Date                          Description
----- ----------------------------- -----------------------------
1     Fri, 04 Oct 2019 11:19:33 CEST some changes
2     Fri, 04 Oct 2019 11:19:52 CEST more changes
[basecm11->softwareimage]% list -r
Name (key)       Path                       Kernel version
---------------- -------------------------- --------------------------
default-image    /cm/images/default-image   3.10.0-957.1.3.el7.x86_64
default-image@1  /cm/images/default-image-1 3.10.0-957.1.3.el7.x86_64
default-image@2  /cm/images/default-image-2 3.10.0-957.1.3.el7.x86_64
[basecm11->softwareimage]% category add storage; commit
```

With the cluster set up like that, the 3 kinds of revision control functionalities in `category` mode can be explained as follows:

1. **Category revision control functionality is defined as unset**
   If the administrator sets the `softwareimage` property for the category to an image without any revision tags:

   Example

   ```
   [basecm11->category]% set storage softwareimage default-image
   ```

   then nodes in the `storage` category take no notice of the revision setting for the image set at category level.

2. **Category revision control sets a specified revision as default**

   If the administrator sets the `softwareimage` property for the category to a specific image, with a revision tag, such as `default-image@1`:

   Example

   ```
   [basecm11->category]% set storage softwareimage default-image@1
   ```

   then nodes in the `storage` category use the image `default-image@1` as their image if nothing is set at node level.

3. **Category revision control sets the latest available revision by default**
   If the administrator sets the `softwareimage` property for the category to a parent image, but tagged with the reserved keyword tag `latest`:

   Example

   ```
   [basecm11->category]% set storage softwareimage default-image@latest
   ```

   then nodes in the `storage` category use the image `default-image@2` if nothing is set at node level. If a new revision of `default-image` is created later on, with a later tag (`@3`, `@4`, `@5`...) then the property takes the new value for the revision, so that nodes in the category will use the new revision as their image.

Revision Control For Images—An Example Session

This section uses a session to illustrate how image revision control is commonly used, with commentary along the way. It assumes a newly installed cluster with Btrfs configured and `/cm/images` migrated as explained in section 14.4.3.

First, a revision of the image is made to save its initial state:

```
[basecm11->softwareimage]% newrevision default-image "Initial state"
```

A new image `default-image@1` is automatically created with a path `/cm/images/default-image-1`. The path is also a subvolume, since it is on a Btrfs partition.

The administrator then makes some modifications to the parent image `/cm/images/default-image`, which can be regarded as a "trunk" revision, for those familiar with SVN or similar revision control systems. For example, the administrator could install some new packages, edit some configuration files, and so on. When finished, a new revision is created by the administrator:

```
[basecm11->softwareimage]% newrevision default-image "Some modifications"
```

This image is then automatically called `default-image@2` and has the path `/cm/images/default-image-2`. If the administrator then wants to test the latest revision on nodes in the `default` category, then this new image can be set at category level, without having to specify it for every node in that category individually:

```
[basecm11->category]% set default softwareimage default-image@2
```

At this point, the content of `default-image` is identical to `default-image@2`. But changes done in `default-image` will not affect what is stored in revision `default-image@2`.

After continuing on with the parent image based on the revision `default-image@2`, the administrator might conclude that the modifications tried are incorrect or not wanted. In that case, the administrator can roll back the state of the parent image `default-image` back to the state that was previously saved as the revision `default-image@1`:

```
[basecm11->softwareimage]% setparent default-image default-image@1
```

The administrator can thus continue experimenting, making new revisions, and trying them out by setting the `softwareimage` property of a category accordingly. Previously created revisions `default-image@1` and `default-image@2` will not be affected by these changes. If the administrator would like to completely purge a specific unused revision, such as `default-image@2` for example, then it can be done with the `-d|--data` option:

```
[basecm11->softwareimage]% remove -d default-image@2
```

The `-d` does a forced recursive removal of the `default-image-2` directory, while a plain `remove` without the `-d` option would simply remove the object from CMDaemon, but leave `default-image-2` alone. This CMDaemon behavior is not unique for Btrfs—it is true for traditional filesystems too. It is however usually very wasteful of storage to do this with non-COW systems.

14.5 BIOS And Firmware Management

14.5.1 Introduction

The main PC BIOS firmware is nowadays a subset of the more general firmware of a system. In older versions of BCM, BIOS and firmware management relied on proprietary vendor implementations. While such legacy implementations are still in use at the time of writing of this section (December 2023), the modern way of managing BIOS and firmware is with the Redfish standard.

The Redfish standard is an industry API standard intended for RESTful management of large numbers of nodes. It is supported by Dell, HPE, Intel, and others.

Redfish uses a pluggable framework architecture with JSON. This makes adding new properties easier, and also makes isolating, debugging, and fixing issues easier.

The CMDaemon front ends of `cmsh` and Base View provide a front end for BIOS and firmware management via Redfish.

14.5.2 BIOS Management With BCM JSON Configuration Templates In Redfish

For BIOS management via Redfish, the JSON configuration is specified per vendor. In BCM the files are kept under:

```
/cm/local/apps/cm-bios-tools/templates/
```

By default, BCM ships with the following configuration file templates:

- `dell_14g.json`
- `dell_15g-amd.json`
- `dell_15g-intel.json`
- `dell_r730.json`
- `hpe_dl110.json`
- `hpe_dl380g10.json`

Example

```
[root@basecm11 ~]# ls -al /cm/local/apps/cm-bios-tools/templates/
total 280
drwxr-xr-x 2 root root    148 Nov 15 08:24 .
drwxr-xr-x 6 root root     64 Nov 15 08:24 ..
-rw-r--r-- 1 root root   5008 Oct 25 19:24 dell_14g.json
```

```
-rw-r--r-- 1 root root   6411 Oct 25 19:24 dell_15g-amd.json
-rw-r--r-- 1 root root  20493 Oct 25 19:24 dell_15g-intel.json
-rw-r--r-- 1 root root   7052 Oct 25 19:24 dell_r730.json
-rw-r--r-- 1 root root 211366 Oct 25 19:24 hpe_dl110.json
-rw-r--r-- 1 root root  24448 Oct 25 19:24 hpe_dl380g10.json
[root@basecm11 templates]# cat dell_r730.json

{
  "displayName": "Dell Inc. - PowerEdge R730",
  "description": "Dell Inc. - PowerEdge R730 - BIOS settings template - v1.0.0",
  "properties": [
    {
      "name": "BootMode",
      "displayName": "Boot Mode",
      "description": "This field determines the boot mode of the system.\n\nSelecting 'UEFI' enables\
      booting to Unified Extensible Firmware Interface (UEFI) capable operating systems.\n\nSelecting\
      'BIOS' (the default) ensures compatibility with operating systems that do not support UEFI.",
      "type": "Enumeration",
      "options": [
        {
          "displayName": "BIOS",
          "value": "Bios"
        },
        {
          "displayName": "UEFI",
          "value": "Uefi"
        }
      ],
      "pos": {
        "g": 0,
        "r": 0,
        "o": 0,
        "w": 6
      }
    },
    {
      "name": "NodeInterleave",
      "displayName": "Node Interleaving",
      "description": "When set to Enabled, memory interleaving is supported if a symmetric memory\
      configuration is installed. When set to Disabled, the system supports Non-Uniform Memory Access\
      (NUMA) (asymmetric) memory configurations.\n\nOperating Systems that are NUMA-aware understand the\
      distribution of memory in a particular system and can intelligently allocate memory in an optimal\
      manner. Operating Systems that are not NUMA aware could allocate memory to a processor that is not\
      local resulting in a loss of performance. Node Interleaving should only be enabled for Operating\
      Systems that are not NUMA aware.\n\nDefault: Disabled",
      "type": "Enumeration",
      "options": [
        {
          "displayName": "Enabled",
          "value": "Enabled"
        },
        {
          "displayName": "Disabled",
          "value": "Disabled"
        }
      ],
```

```
      "pos": {
        "g": 0,
        "r": 0,
        "o": 1,
        "w": 6
      }
    },
    {
      "name": "SnoopMode",
      "displayName": "Snoop Mode",
      "description": "Allows tuning of memory performances under different memory bandwidths. The optimal\
      Snoop Mode setting is highly dependent on workload type.\n\nEarly Snoop is best used for latency sensitive\
      workloads. This setting offers the best balance between workload effects.\n\nHome Snoop is best used for\
      NUMA workloads that need maximum local and remote memory bandwidth.\n\nCluster on Die is best used for\
      highly NUMA optimized workloads. This setting offers the best case local memory latency, but worst case\
      remote latency.\n\nCluster On Die is only available when Node Interleaving is Disabled.\n\nOpportunistic\
      Snoop Broadcast, available on select processor models, works well for workloads of mixed NUMA optimization.\
      It offers a good balance of latency and bandwidth.\n\nDefault: Early Snoop",
      "type": "Enumeration",
      "options": [
        {
          "displayName": "Early Snoop",
          "value": "EarlySnoop"
        },
...
```

BCM BIOS Configuration States And Operations Overview

In BCM, a BIOS configuration of a node or category can be thought of as being in one of 4 possible states, with 3 possible operations that apply the changes to the states. This is shown by the following schematic:

```
cmsh or          commit                          bios apply                        reboot
Base View  --------------> CMDaemon database -----------> BIOS (pending) -----------> BIOS (live)
```

For example, the cluster administrator might adjust the BIOS configuration for the node or category in `cmsh`. The state set within `cmsh` then becomes a state stored within the CMDaemon database after the `commit` operation of `cmsh` is carried out.

The cluster administrator can then apply the BIOS configuration that is stored in the CMDaemon database by running the `bios apply` operation from within `cmsh`. The BIOS configuration is then taken up as the `BIOS (pending)` state stored in the BIOS firmware of the node (or category).

Finally, the cluster administrator can implement the BIOS, so that it runs on the live node (or category). This happens when carrying out a `reboot` operation for that node (or category). The BIOS configuration that was a pending BIOS setting then becomes a live BIOS setting.

The details of how these changes can be carried out are explained in the following sections.

Example BIOS Configuration Session In `cmsh`

In `cmsh`, the BIOS settings can be viewed, compared, and applied at the `device` mode level or `category` mode level.

The BIOS settings for the various states can alternatively be managed using the `cm-bios-manage` utility (page 752). However the `cmsh` or Base View front ends to `cm-bios-manage` are easier to use.

Model: The model must be set for the BIOS settings before other BIOS settings can be managed. If it is not set, then the `status` command in `biossettings` displays an error, as indicated by the following `cmsh` session:

Example

```
[basecm11->device[node002]->biossettings]% status
Parameter                       Configured Pending Live
------------------------------- ---------- ------- ----
Pending errors:
No model defined
```

It can be set with the help of tab-completion:

Example

```
[basecm11->device[node002]->biossettings]% set model<tab><tab>
dell_r730  hpe_dl380
[basecm11->device[node002]->biossettings]% set model hpe_dl380
[basecm11->device*[node002*]->biossettings*]% commit
```

Viewing BIOS parameters: Each BIOS parameter can now have its value listed and compared by state.

The `status` command shows a list of the parameters, and their values are displayed for each state. So, the state columns show:

1. the BIOS parameter as stored in the CMDaemon database (the `Configured` column),
2. the BIOS parameter as stored on the node itself (the `Pending` column),
3. the BIOS parameter as implemented on the node itself (the `Live` column)

Example

```
[basecm11->device[node002]]% biossettings
[basecm11->device[node002]->biossettings]% status
Parameter                                    Configured     Pending Live
-------------------------------------------- -------------- ------- ----------------------------
High Precision Event Timer (HPET) ACPI Support  < default >    -       Enabled
Adjacent Sector Prefetch                        < default >    -       Enabled
Boot Mode                                       < default >    -       UEFI Mode
Boot Order Policy                               < default >    -       Retry Boot Order Indefinitely
Channel Interleaving                            < default >    -       Enabled
Collaborative Power Control                     < default >    -       Enabled
Consistent Device Naming                        < default >    -       CDN Support for LOMs and Slots
Custom POST Message                             < default >    -
LLC Prefetch                                    < default >    -       Disabled
Local/Remote Threshold                          < default >    -       Auto
Maximum Memory Bus Frequency                    < default >    -       Auto
Maximum PCI Express Speed                       < default >    -       Per Port Control
Memory Mirroring Mode                           < default >    -       Full Mirror
Memory Patrol Scrubbing                         < default >    -       Enabled
Memory Refresh Rate                             < default >    -       1x Refresh
Minimum Processor Idle Power Package C-State    < default >    -       Package C6 (retention) State
Minimum Processor Idle Power Core C-State       < default >    -       C6 State
Mixed Power Supply Reporting                    < default >    -       Enabled
Network Boot Retry Support                      < default >    -       Enabled
Node Interleaving                               < default >    -       Disabled
NUMA Group Size Optimization                    < default >    -       Flat
Embedded NVM Express Option ROM                 < default >    -       Enabled
NVMe PCIe Resource Padding                      < default >    -       Normal
Persistent Memory Address Range Scrub           < default >    -       Enabled
```

```
POST Verbose Boot Progress                    < default >    -       Disabled
Power-On Delay                                < default >    -       No Delay
Server Asset Tag                              < default >    -
Network Boot Retry Count                      < default >    -       20
```

In the preceding example, each parameter of the configured column has a setting of `< default >`. This means that the value for the configured setting is a null value, as achieved by running the `clear` command for that setting. A BIOS setting configured with `< default >` as a value does nothing based on that configuration setting when doing BIOS management operations. Thus, for example, the setting for `Boot Mode` in the `Configured` column can only be made to cause a change in the `Pending` or `Live` columns if it takes a value that is not `default`.

Changing and checking changes for BIOS parameters: Thus, if the states for a node are as follows for the `Boot Mode` parameter:

Example

```
[basecm11->device[node002]->biossettings]% status |head -2 ; status |grep "Boot Mode"
Parameter                                     Configured       Pending         Live
--------------------------------------------- ---------------- --------------- ----------------
Boot Mode                                     < default >      -               UEFI Mode
```

then have the `Live` state value change from `UEFI Mode` to `Legacy BIOS Mode`:

1. the first step is to change the `Configured` state:

   Example

   ```
   [basecm11->device[node002]->biossettings]% set boot mode<tab><tab>
   legacy bios mode  uefi mode
   [basecm11->device[node002]->biossettings]% set boot mode legacy bios mode
   [basecm11->device*[node002]->biossettings*]% commit
   [basecm11->device[node002]->biossettings]% status |head -2 ; status |grep "Boot Mode"
   Parameter                                     Configured       Pending         Live
   --------------------------------------------- ---------------- --------------- ----------------
   Boot Mode                                     Legacy BIOS Mode -               UEFI Mode
   ```

   - Beside using the `status` command within the `biossettings` submode, the existing BIOS states that are configured (in CMDaemon) and detected (on the live node) can also be checked with the `bios check` command at `node` or `category` level:

     Example

     ```
     [basecm11->device[node002]->biossettings]% ..
     [basecm11->device[node002]]% bios check
     Result     Parameter                      Configured                      Detected
     ---------- ------------------------------ ------------------------------- ----------
     different  BootMode                       LegacyBios                      Uefi
     ```

     The `bios check` command shows a result if there is a difference between the configured (CMDaemon) and detected (live) configuration.

2. The next step is to apply the configuration change to the `Pending` BIOS state:

   Example

```
[basecm11->device[node002]]% bios apply
Node             Result   Output                          Error
---------------- -------- ------------------------------- -------------------------------
node002          good
[basecm11->device[node002]]% biossettings
[basecm11->device[node002]->biossettings]% status |head -2 ; status |grep "Boot Mode"
Parameter                                      Configured       Pending          Live
---------------------------------------------- ---------------- ---------------- ---------------
Boot Mode                                      Legacy BIOS Mode Legacy BIOS Mode UEFI Mode
```

3. Finally, a reboot causes the Pending value to be made live:

Example

```
[basecm11->device[node002]->biossettings]% ..
[basecm11->device[node002]]% reboot
[cluster administrator waits for the node to finish rebooting]
```

The BIOS change to the Live state is then complete. The Pending and Configured state values are cleared automatically too, so that the states for the Boot Mode parameter now show:

Example

```
[basecm11->device[node002]->biossettings]% status |head -2 ; status |grep "Boot Mode"
Parameter                                      Configured       Pending         Live
---------------------------------------------- ---------------- --------------- ----------------
Boot Mode                                      < default >      -               Legacy BIOS Mode
```

BIOS Configuration Via cm-bios-manage

This section can usually be skipped because the administrator is not expected to use the cm-bios-manage utility directly.

This is because it is relatively low-level, and because the easiest way for a cluster administrator to manage the BIOS of a cluster via Redfish is usually via the cmsh (page 749) or Base View front ends.

The cm-bios-manage help text is:

```
[root@basecm11 ~]# /cm/local/apps/cm-bios-tools/bin/cm-bios-manage -h
usage: cm-bios-manage [-h]
                      [-a | -c | -f VENDOR_MODEL | -p VENDOR_MODEL | -P VENDOR_MODEL PROFILE | -t |
-T VENDOR_MODEL] [-l] [-d]

Script used by cmd to manage BIOS settings.

optional arguments:
  -h, --help            show this help message and exit
  -a, --apply           apply settings.
  -c, --check           check defined settings.
  -f VENDOR_MODEL, --fetch VENDOR_MODEL
                        fetch settings based on JSON template of specified model.
  -p VENDOR_MODEL, --profiles VENDOR_MODEL
                        list all profiles for the specified model.
  -P VENDOR_MODEL PROFILE, --profile VENDOR_MODEL PROFILE
                        display profile for the specified model and specified profile.
  -t, -m, --vendor-types, --models
                list all supported HW models.
  -T VENDOR_MODEL, --template VENDOR_MODEL
```

```
                        display JSON template of specified HW model.
  -l, --live            fetch live settings, instead of pending settings.
                        used with --fetch and --check options.
  -d, --debug           enable debug messages.
```

A session with options might run as follows:

Example

```
[root@basecm11 ~]# /cm/local/apps/cm-bios-tools/python/cm-bios-manage -t
[
  "dell_r730",
  "hpe_dl380"
]
[root@basecm11 ~]# /cm/local/apps/cm-bios-tools/python/cm-bios-manage -p hpe_dl380
[
  "test"
]
[root@basecm11 ~]# /cm/local/apps/cm-bios-tools/python/cm-bios-manage -P hpe_dl380 test
{
  "test": "xyz"
}
[root@basecm11 ~]# /cm/local/apps/cm-bios-tools/python/cm-bios-manage -T hpe_dl380
[
  {
    "displayName": "High Precision Event Timer (HPET) ACPI Support",
    "name": "AcpiHpet",
    "pos": {
      "w": 12,
      "r": 0,
      "o": 0,
      "g": 0
    },
...
```

14.5.3 Updating BIOS And Firmware Versions

There are two ways that the firmware can be updated. A legacy way based on DOS tools (page 753), and a more recent way, based on CMDaemon and Redfish (page 754).

Updating A BIOS Via DOS Tools

The legacy way of upgrading a BIOS to a new version involves using the DOS tools that were supplied with the BIOS to flash a new BIOS. The flash tool and the BIOS image must be copied to a DOS image. The file `autoexec.bat` should be altered to invoke the flash utility with the correct parameters. In case of doubt, it can be useful to boot the DOS image and invoke the BIOS flash tool manually. Once the correct parameters have been determined, they can be added to the `autoexec.bat`.

After a BIOS upgrade, the contents of the NVRAM may no longer represent a valid BIOS configuration because different BIOS versions may store a configuration in different formats. It is therefore recommended to also write updated NVRAM settings immediately after flashing a BIOS image.

The next section describes how to boot the DOS image.

Booting the DOS image: To boot the DOS image over the network, it first needs to be copied to software image's `/boot` directory, and must be world-readable.

Example

```
cp flash.img /cm/images/default-image/boot/bios/flash.img
chmod 644 /cm/images/default-image/boot/bios/flash.img
```

An entry is added to the PXE boot menu to allow the DOS image to be selected. This can easily be achieved by modifying the contents of `/cm/images/default-image/boot/bios/menu.conf`, which is by default included automatically in the PXE menu. By default, one entry `Example` is included in the PXE menu, which is however invisible as a result of the `MENU HIDE` option. Removing the `MENU HIDE` line will make the BIOS flash option selectable. Optionally the `LABEL` and `MENU LABEL` may be set to an appropriate description.

The option `MENU DEFAULT` may be added to make the BIOS flash image the default boot option. This is convenient when flashing the BIOS of many nodes.

Example

```
LABEL FLASHBIOS
  KERNEL memdisk
  APPEND initrd=bios/flash.img
  MENU LABEL ^Flash BIOS
#  MENU HIDE
  MENU DEFAULT
```

The `bios/menu.conf` file may contain multiple entries corresponding to several DOS images to allow for flashing of multiple BIOS versions or configurations.

Firmware Configuration And Updates Via CMDaemon

For systems that support the Redfish protocol, such as HPE iLO and DGX H100, firmware management requires setting `firmwaremanagemode`, within a `bmcsettings` submode. HPE systems including DL380a Gen 12 are supported identically to earlier HPE generations under this same Redfish-based firmware management, with no separate procedure required.

In addition, for DGX hardware, after the firmware has been flashed over to it, an activation based on an AC power cycle is required. Details on this are given (page 761) as part of the DGX installation example later on (page 756).

Setting `firmwaremanagemode`: The value of `firmwaremanagemode` is selected appropriately by the administrator according to the node hardware:

Example

```
[basecm11->device*[node001*]->bmcsettings*]% set firmwaremanagemode <TAB><TAB>
auto      b200      b300      gb200     gb200sw   gb300     gb300sw   h100      iLO       none
```

The `bmcsettings` submode can be accessed and set within an instance of the `device`, `category`, or `partition` modes.

Running `firmware` operations and options: After `firmwaremanagemode` has been set, the `firmware` command can be run under the `device` mode of `cmsh` to carry out Redfish protocol updates. Firmware updated via Redfish need not be just the PC main system BIOS, but can also be the flashable software of subsystems, for example: NICs.

The `help firmware` command provides a help page covering the operations and options for the `firmware` command. Some of them are described next:

- The `firmware` command includes the following operations:
  - `info`: provides information on the firmwares available on the head node, available for uploading to nodes. By default these are files that the cluster administrator has picked up from the vendor and has placed under `/cm/local/apps/cmd/etc/htdocs/bios/firmware/`.

- `list`: (only for iLO) provides a list of firmware states on specified nodes.
- `upload`: (only for iLO) takes a specified firmware from the files listed by the `info` option, and copies it over to the specified nodes. For an HPE iLO system, the upload is carried out to a special flash storage, only visible to the BIOS and Redfish queries. After it is in that special flash location, it can be flashed to where the firmware is actually run.
- `remove`: (only for iLO) removes a specified firmware from the special flash storage of specified nodes
- `flash` : carries out the flashing of the firmware from the flash storage to the location where the firmware runs, for the specified nodes
- `status`: shows the status of the flash operation. A table is displayed with rows listing the component, version, and state, among other items. The `State` column of the output can show the following values:
  - `pending`: the flash operation is pending
  - `exception`: the flash operation failed during execution
  - `flashing`: the flash operation is being executed
  - `completed`: the flash operation succeeded
  - `current`: the firmware of the listed component with the listed version is activated

- The `firmware` command also includes the following options:
  - `--targets`: (only for DGX hosts) specifies names for particular component firmware targets. Without this option, the default is that updates are carried out automatically only for newer firmware components from the firmware package files. Overriding the default should not be needed, and is typically not recommended. Specifying `--targets list -v` lists the possible components.
  - `--force`: (only for DGX hosts) needed to carry out a downgrade, and also needed for some other DGX firmware cases as described in `https://docs.nvidia.com/dgx/dgxh100-fw-update-guide/sequence.html#update-steps`
  - `--dry-run`: (only for DGX hosts) pretends to carry out an installation, so that the administrator can get an idea of what components are affected from the output of the mock installation run

An HP iLO5 firmware upgrade example: The following session shows `node001` getting uploaded and flashed with a firmware, and then having the firmware removed.

Example

```
[basecm11->device]% firmware info
Device   Filename     Component        Version  State      Progress Result   Size     Date
-------- ------------ ---------------- -------- ---------- -------- -------- -------- --------------------
basecm11 iLO5-2.42    iLO5             2.42     undefined  N/A              8.6MiB   02/28/2022, 11:22:55
basecm11 iLO5-2.43    iLO5             2.43     undefined  N/A              8.7MiB   02/28/2022, 11:22:55
basecm11 iLO5-2.44    iLO5             2.44     undefined  N/A              8.8MiB   02/28/2022, 11:22:55
basecm11 iLO5-2.45    iLO5             2.45     undefined  N/A              8.9MiB   02/28/2022, 11:22:55

[basecm11->device]% firmware list -n node001

[basecm11->device]% firmware upload iLO5-2.42 -n node001
Device   Result   Output                 Error
-------- -------- ---------------------- --------------------------------
node001  good     uploading: iLO5-2.42
```

```
[basecm11->device]% firmware list -n node001
Device   Filename     Component        Version  State      Progress Result   Size     Date
-------- ------------ ---------------- -------- ---------- -------- -------- -------- --------------------
node001  iLO5-2.42    iLO5             2.42     completed  N/A              8.6MiB   02/28/2022, 11:23:31

[basecm11->device]% firmware flash iLO5-2.42 -n node001
Device   Result   Output                 Error
-------- -------- ---------------------- --------------------------------
node001  good     flashing: iLO5-2.42

[basecm11->device]% firmware status -n node001
Device   Filename     Component        Version  State     Progress Result   Size     Date
-------- ------------ ---------------- -------- --------- -------- -------- -------- --------------------
node001               iLO5             2.42     flashing  35.6%            N/A      02/28/2022, 11:27:27

[basecm11->device]% firmware status -n node001
Device   Filename     Component        Version  State      Progress Result   Size     Date
-------- ------------ ---------------- -------- ---------- -------- -------- -------- --------------------
node001               iLO5             2.42     completed  N/A              N/A      02/28/2022, 11:27:27

[basecm11->device]% firmware remove iLO5-2.42 -n node001
Device   Result   Output                 Error
-------- -------- ---------------------- --------------------------------
node001  good     removed: iLO5-2.42

[basecm11->device]% firmware list -n node001
```

A GPU tray upgrade example on the DGX H100: The procedure to upgrade DGX firmware varies slightly across the different hardwares (H100, B200, GB200 ...). For the DGX H100 the upgrade procedure is as follows:

Obtaining and placing the firmware packages on the DGX H100 cluster: Firmware for the DGX H100 is available for the motherboard tray (chassis) components, and for the GPU tray components.

Firmware packages for the DGX H100 can be obtained via the DGX support portal, which can be reached from:

`https://docs.nvidia.com/dgx/dgxh100-fw-update-guide/about.html#firmware-update-prerequisites`.

The packages are available as `.fwpkg` packages. They should be placed in the head node directory at `/cm/local/apps/cmd/etc/htdocs/bios/firmware/`

under the appropriate existing subdirectory for the platform type:

Example

```
[basecm11 ~]# ls -l /cm/local/apps/cmd/etc/htdocs/bios/firmware/h100
-rw-r--r-- 1 root root 135723563 Dec 12 14:27 nvfw_DGX-H100_0003_230817.1.1_custom_prod-signed.fwpkg
-rw-r--r-- 1 root root 135723563 Dec 12 14:27 nvfw_DGX-H100_0003_230905.1.0_custom_prod-signed.fwpkg
-rw-r--r-- 1 root root 135723563 Dec 12 14:27 nvfw_DGX-H100_0003_230920.1.0_custom_prod-signed.fwpkg
-rw-r--r-- 1 root root 106091440 Dec 12 14:27 nvfw_DGX-HGX-H100x8_0002_230705.1.1_prod-signed.fwpkg
```

Configuring BMC settings in BCM for the DGX H100: The BMC interface and settings should be configured for the DGX H100. Typically this requires:

- adding a BMC network (section 3.2.2)

- configuring its BMC settings (section 3.7.2) to be able to carry out the Redfish protocol. This means setting the:
  - username
  - userid
  - password
  - firmware mode for Redfish
- adding an interface to the network for the node (section 3.7.1).

Example

```
[basecm11 ~]# cmsh
[basecm11]% network
[basecm11->network]% add bmcnet
[basecm11->network*[bmcnet*]]% set baseaddress 10.148.0.0
[basecm11->network*[bmcnet*]]% set domainname bmc.cluster
[basecm11->network*[bmcnet*]]% commit
[basecm11->network[bmcnet]]% partition
[basecm11->partition[base]]% bmcsettings
[basecm11->partition[base]->bmcsettings]% set username admin
[basecm11->partition*[base*]->bmcsettings*]% set userid 0
[basecm11->partition*[base*]->bmcsettings*]% set password < password>
[basecm11->partition*[base*]->bmcsettings*]% set firmwaremanagemode <TAB><TAB>
auto     b200     b300     gb200    gb200sw  gb300    gb300sw  h100     iL0      none
[basecm11->partition*[base*]->bmcsettings*]% set firmwaremanagemode h100
[basecm11->partition*[base*]->bmcsettings*]% commit
[basecm11->partition[base]->bmcsettings]% device
[basecm11->device]% interfaces node001
[basecm11->device[node001]->interfaces]% add bmc
[basecm11->device[node001]->interfaces]% add bmc ipmi0
[basecm11->device*[node001*]->interfaces*[ipmi0*]]% set network bmcnet
[basecm11->device*[node001*]->interfaces*[ipmi0*]]% set ip 10.148.0.1
```

Managing, installing and updating the firmware package in BCM for the DGX H100: BCM can then display information about the files in that head node firmware directory with the `firmware info` command:

Example

```
[basecm11->device]% firmware info
Device   Filename                       Component         Version             State      Progress  ...
-------- ---------------------------- ----------------- ------------------- ---------- --------
basecm11 nvfw_DGX-H100_...fwpkg        DGX-H100-Chassis  DGX-H100_0003_2...  available  N/A
basecm11 nvfw_DGX-H100_...fwpkg        DGX-H100-Chassis  DGX-H100_0003_2...  available  N/A
basecm11 nvfw_DGX-HGX-H100x8_...fwpkg  DGX-H100-GPU      DGX-HGX-H100x8_...  available  N/A
```

The `firmware status` command displays information on the state of the firmware components running on the node:

Example

```
[basecm11->device]% firmware status -n node001
Device  Filename Component                Version              State   Progress  Result Size Date
------- -------- ----------------------- -------------------- ------- --------- ------ ---- ----
```

```
node001          CPLDMB_0                 0.2.1.0              current  N/A             N/A
node001          CPLDMID_0                0.2.1.0              current  N/A             N/A
node001          EROT_BIOS_0              00.04.0020.0000_n00  current  N/A             N/A
node001          EROT_BMC_0               00.04.0020.0000_n00  current  N/A             N/A
node001          HGX_FW_BMC_0             HGX-22.10-1-rc1      current  N/A             N/A
node001          HGX_FW_ERoT_BMC_0        00.02.0100.0000_n00  current  N/A             N/A
node001          HGX_FW_ERoT_FPGA_0       00.02.0100.0000_n00  current  N/A             N/A
node001          HGX_FW_ERoT_NVSwitch_0   00.02.0100.0000_n00  current  N/A             N/A
node001          HGX_FW_ERoT_NVSwitch_1   00.02.0100.0000_n00  current  N/A             N/A
node001          HGX_FW_ERoT_NVSwitch_2   00.02.0100.0000_n00  current  N/A             N/A
node001          HGX_FW_ERoT_NVSwitch_3   00.02.0100.0000_n00  current  N/A             N/A
node001          HGX_FW_ERoT_PCIeSwitch_0 00.02.0100.0000_n00  current  N/A             N/A
node001          HGX_FW_FPGA_0            2.0A                 current  N/A             N/A
node001          HGX_FW_GPU_SXM_1         96.00.70.00.01       current  N/A             N/A
node001          HGX_FW_GPU_SXM_2         96.00.70.00.01       current  N/A             N/A
node001          HGX_FW_GPU_SXM_3         96.00.70.00.01       current  N/A             N/A
node001          HGX_FW_GPU_SXM_4         96.00.70.00.01       current  N/A             N/A
node001          HGX_FW_GPU_SXM_5         96.00.70.00.01       current  N/A             N/A
node001          HGX_FW_GPU_SXM_6         96.00.70.00.01       current  N/A             N/A
node001          HGX_FW_GPU_SXM_7         96.00.70.00.01       current  N/A             N/A
node001          HGX_FW_GPU_SXM_8         96.00.70.00.01       current  N/A             N/A
node001          HGX_FW_NVSwitch_0        96.00.3F.00.01       current  N/A             N/A
node001          HGX_FW_NVSwitch_1        96.00.3F.00.01       current  N/A             N/A
node001          HGX_FW_NVSwitch_2        96.00.3F.00.01       current  N/A             N/A
node001          HGX_FW_NVSwitch_3        96.00.3F.00.01       current  N/A             N/A
node001          HGX_FW_PCIeRetimer_0     2.7.0                current  N/A             N/A
node001          HGX_FW_PCIeRetimer_1     2.7.0                current  N/A             N/A
node001          HGX_FW_PCIeRetimer_2     2.7.0                current  N/A             N/A
node001          HGX_FW_PCIeRetimer_3     2.7.0                current  N/A             N/A
node001          HGX_FW_PCIeRetimer_4     2.7.0                current  N/A             N/A
node001          HGX_FW_PCIeRetimer_5     2.7.0                current  N/A             N/A
node001          HGX_FW_PCIeRetimer_6     2.7.0                current  N/A             N/A
node001          HGX_FW_PCIeRetimer_7     2.7.0                current  N/A             N/A
node001          HGX_FW_PCIeSwitch_0      1.7.5A               current  N/A             N/A
node001          HostBIOS_0               01.00.00             current  N/A             N/A
node001          HostBMC_0                23.00.00             current  N/A             N/A
node001          PCIeRetimer_0            1.30.0               current  N/A             N/A
node001          PCIeRetimer_1            1.30.0               current  N/A             N/A
node001          PCIeSwitch_0             0.0.1                current  N/A             N/A
node001          PCIeSwitch_1             1.0.1                current  N/A             N/A
node001          PSU_0                    0202.0200.0200       current  N/A             N/A
node001          PSU_1                    0202.0200.0200       current  N/A             N/A
node001          PSU_2                    0202.0200.0200       current  N/A             N/A
node001          PSU_3                    0202.0201.0202       current  N/A             N/A
node001          PSU_4                    0202.0201.0203       current  N/A             N/A
node001          PSU_5                    0202.0200.0200       current  N/A             N/A
```

Components can be updated by installing associated firmware packages with the `firmware flash` command. For example, the GPU tray firmware can be installed with one of the `nvfw_dgx-hgx-h100x8*` packages:

Example

```
[basecm11->device[node001]]% firmware flash <TAB><TAB>
nvfw_dgx-h100_0003_230817.1.1.fwpkg
nvfw_dgx-h100_0003_230920.1.0.fwpkg
```

```
nvfw_dgx-hgx-h100x8_0002_230705.1.1.fwpkg
[basecm11->device[node001]]% firmware flash nvfw_dgx-hgx-h100x8_0002_230705.1.1.fwpkg
Device           flashing                                  Result   Error
---------------- ----------------------------------------- -------- ----------
node001          nvfw_DGX-HGX-H100x8_0002_230705.1.1.fwpkg  good
```

The `firmware status` command then shows the installation progress for the GPU tray components during flashing:

Example

```
[basecm11->device]% firmware -n node001 status
Device  Filename             Component                 Version              State     Progress...
------- -------------------- ------------------------- -------------------- --------- --------
node001                      CPLDMB_0                  0.2.1.0              current   N/A
node001                      CPLDMID_0                 0.2.1.0              current   N/A
node001                      EROT_BIOS_0               00.04.0020.0000_n00  current   N/A
node001                      EROT_BMC_0                00.04.0020.0000_n00  current   N/A
node001                      HostBIOS_0                01.00.00             current   N/A
node001                      HostBMC_0                 23.00.00             current   N/A
node001                      PCIeRetimer_0             1.30.0               current   N/A
node001                      PCIeRetimer_1             1.30.0               current   N/A
node001                      PCIeSwitch_0              0.0.1                current   N/A
node001                      PCIeSwitch_1              1.0.1                current   N/A
node001                      PSU_0                     0202.0200.0200       current   N/A
node001                      PSU_1                     0202.0200.0200       current   N/A
node001                      PSU_2                     0202.0200.0200       current   N/A
node001                      PSU_3                     0202.0201.0202       current   N/A
node001                      PSU_4                     0202.0201.0203       current   N/A
node001                      PSU_5                     0202.0200.0200       current   N/A
node001 nvfw_DGX-HG...fwpkg  HGX_FW_BMC_0              HGX-22.10-1-rc1      flashing  39.1%
node001 nvfw_DGX-HG...fwpkg  HGX_FW_ERoT_BMC_0         00.02.0100.0000_n00  flashing  39.1%
node001 nvfw_DGX-HG...fwpkg  HGX_FW_ERoT_FPGA_0        00.02.0100.0000_n00  flashing  39.1%
node001 nvfw_DGX-HG...fwpkg  HGX_FW_ERoT_NVSwitch_0    00.02.0100.0000_n00  flashing  39.1%
node001 nvfw_DGX-HG...fwpkg  HGX_FW_ERoT_NVSwitch_1    00.02.0100.0000_n00  flashing  39.1%
node001 nvfw_DGX-HG...fwpkg  HGX_FW_ERoT_NVSwitch_2    00.02.0100.0000_n00  flashing  39.1%
node001 nvfw_DGX-HG...fwpkg  HGX_FW_ERoT_NVSwitch_3    00.02.0100.0000_n00  flashing  39.1%
node001 nvfw_DGX-HG...fwpkg  HGX_FW_ERoT_PCIeSwitch_0  00.02.0100.0000_n00  flashing  39.1%
node001 nvfw_DGX-HG...fwpkg  HGX_FW_FPGA_0             2.0A                 flashing  39.1%
node001 nvfw_DGX-HG...fwpkg  HGX_FW_GPU_SXM_1         96.00.70.00.01       flashing  39.1%
node001 nvfw_DGX-HG...fwpkg  HGX_FW_GPU_SXM_2         96.00.70.00.01       flashing  39.1%
node001 nvfw_DGX-HG...fwpkg  HGX_FW_GPU_SXM_3         96.00.70.00.01       flashing  39.1%
node001 nvfw_DGX-HG...fwpkg  HGX_FW_GPU_SXM_4         96.00.70.00.01       flashing  39.1%
node001 nvfw_DGX-HG...fwpkg  HGX_FW_GPU_SXM_5         96.00.70.00.01       flashing  39.1%
node001 nvfw_DGX-HG...fwpkg  HGX_FW_GPU_SXM_6         96.00.70.00.01       flashing  39.1%
node001 nvfw_DGX-HG...fwpkg  HGX_FW_GPU_SXM_7         96.00.70.00.01       flashing  39.1%
node001 nvfw_DGX-HG...fwpkg  HGX_FW_GPU_SXM_8         96.00.70.00.01       flashing  39.1%
node001 nvfw_DGX-HG...fwpkg  HGX_FW_NVSwitch_0        96.00.3F.00.01       flashing  39.1%
node001 nvfw_DGX-HG...fwpkg  HGX_FW_NVSwitch_1        96.00.3F.00.01       flashing  39.1%
node001 nvfw_DGX-HG...fwpkg  HGX_FW_NVSwitch_2        96.00.3F.00.01       flashing  39.1%
node001 nvfw_DGX-HG...fwpkg  HGX_FW_NVSwitch_3        96.00.3F.00.01       flashing  39.1%
node001 nvfw_DGX-HG...fwpkg  HGX_FW_PCIeRetimer_0     2.7.0                flashing  39.1%
node001 nvfw_DGX-HG...fwpkg  HGX_FW_PCIeRetimer_1     2.7.0                flashing  39.1%
node001 nvfw_DGX-HG...fwpkg  HGX_FW_PCIeRetimer_2     2.7.0                flashing  39.1%
node001 nvfw_DGX-HG...fwpkg  HGX_FW_PCIeRetimer_3     2.7.0                flashing  39.1%
node001 nvfw_DGX-HG...fwpkg  HGX_FW_PCIeRetimer_4     2.7.0                flashing  39.1%
```

```
node001 nvfw_DGX-HG...fwpkg  HGX_FW_PCIeRetimer_5      2.7.0              flashing  39.1%
node001 nvfw_DGX-HG...fwpkg  HGX_FW_PCIeRetimer_6      2.7.0              flashing  39.1%
node001 nvfw_DGX-HG...fwpkg  HGX_FW_PCIeRetimer_7      2.7.0              flashing  39.1%
node001 nvfw_DGX-HG...fwpkg  HGX_FW_PCIeSwitch_0       1.7.5A             flashing  39.1%
```

The non-GPU-related firmware components are not updated.

If the `firmware flash` command stage has completed its run, then when the `firmware status` command is run, the `Component` and `Version` columns show output indicating that the firmware version is transitioning. The rows that have to do with the GPU firmware status are the rows with the string `nvfw_DGX-HG` in the `Filename` column. Those GPU firmware rows have `Component` and `Version` column entries such as:

Example

```
[basecm11->device[node001]]% firmware status| head -2| cut -b66-133; firmware status| grep nvfw| cut -b66-133
Component                 Version
------------------------- ------------------------------------------
HGX_FW_BMC_0              HGX-22.10-1-rc1 -> HGX-22.10-1-rc44
HGX_FW_ERoT_BMC_0         00.02.0100.0000_n00 -> 00.02.0134.0000_n00
HGX_FW_ERoT_FPGA_0        00.02.0100.0000_n00 -> 00.02.0134.0000_n00
HGX_FW_ERoT_NVSwitch_0    00.02.0100.0000_n00 -> 00.02.0134.0000_n00
HGX_FW_ERoT_NVSwitch_1    00.02.0100.0000_n00 -> 00.02.0134.0000_n00
HGX_FW_ERoT_NVSwitch_2    00.02.0100.0000_n00 -> 00.02.0134.0000_n00
HGX_FW_ERoT_NVSwitch_3    00.02.0100.0000_n00 -> 00.02.0134.0000_n00
HGX_FW_ERoT_PCIeSwitch_0  00.02.0100.0000_n00 -> 00.02.0134.0000_n00
HGX_FW_FPGA_0             2.0A -> 2.2C
HGX_FW_GPU_SXM_1          96.00.70.00.01 -> 96.00.74.00.01
HGX_FW_GPU_SXM_2          96.00.70.00.01 -> 96.00.74.00.01
HGX_FW_GPU_SXM_3          96.00.70.00.01 -> 96.00.74.00.01
HGX_FW_GPU_SXM_4          96.00.70.00.01 -> 96.00.74.00.01
HGX_FW_GPU_SXM_5          96.00.70.00.01 -> 96.00.74.00.01
HGX_FW_GPU_SXM_6          96.00.70.00.01 -> 96.00.74.00.01
HGX_FW_GPU_SXM_7          96.00.70.00.01 -> 96.00.74.00.01
HGX_FW_GPU_SXM_8          96.00.70.00.01 -> 96.00.74.00.01
HGX_FW_NVSwitch_0         96.00.3F.00.01 -> 96.10.3F.00.01
HGX_FW_NVSwitch_1         96.00.3F.00.01 -> 96.10.3F.00.01
HGX_FW_NVSwitch_2         96.00.3F.00.01 -> 96.10.3F.00.01
HGX_FW_NVSwitch_3         96.00.3F.00.01 -> 96.10.3F.00.01
HGX_FW_PCIeRetimer_0      2.7.0 -> 2.7.9
HGX_FW_PCIeRetimer_1      2.7.0 -> 2.7.9
HGX_FW_PCIeRetimer_2      2.7.0 -> 2.7.9
HGX_FW_PCIeRetimer_3      2.7.0 -> 2.7.9
HGX_FW_PCIeRetimer_4      2.7.0 -> 2.7.9
HGX_FW_PCIeRetimer_5      2.7.0 -> 2.7.9
HGX_FW_PCIeRetimer_6      2.7.0 -> 2.7.9
HGX_FW_PCIeRetimer_7      2.7.0 -> 2.7.9
HGX_FW_PCIeSwitch_0       1.7.5A -> 1.7.5F
```

In the preceding, the `grep` command is to select just the GPU-related rows, and the `cut` commands are used to remove some columns, to make the output clearer to the reader.

The `firmware status` GPU firmware rows also have the following output columns:

- the `State` column, which shows `Pending` if there are activation steps still required to complete the firmware update
- the `Result` column, which suggests the recommended action to take to activate the firmware

Example

```
[basecm11->device[node001]]% firmware status   #other rows and columns omitted for readability
State    Progress Result                                Size     Date
-------- -------- ------------------------------------ -------- --------
pending  N/A      success: AC power cycle to activate  1.22KiB
pending  N/A      success: AC power cycle to activate  1.22KiB
pending  N/A      success: AC power cycle to activate  1.22KiB
...
```

AC and DC power cycling for the DGX H100: As suggested in the `Result` column in the preceding example, an `AC power cycle` must then be carried out on the node to activate the GPU firmware with the new versions. The `AC power cycle` tag is a part of Redfish terminology, and it implies that all the AC power inputs need to be cut off. This can be carried out via a PDU powering off the entire system, or it can be done physically, by hand, by pulling out all the mains (AC) power leads to the system.

A regular `power reset` command run from the `device` mode of `cmsh`, which is a board level (DC) power cycle by default, is not enough to initialize some of the components for the DGX H100.

Details on firmware activation for the DGX H100 can be found at `https://docs.nvidia.com/dgx/dgxh100-fw-update-guide/about.html#firmware-update-activation`.

Successful activation and completed state confirmations for the DGX H100: After the power cycle, the output from `firmware status` indicates that the firmware now active on the node matches that of the `.fwpkg` file version.

It does this by displaying a value of

`success: activated`

in the `Result` column.

The output also now shows the new version value for the firmware component in the `Version` column, and indicates the firmware transition is now over with a value of `completed` in the `State` column:

```
[basecm11->device]% firmware -n node001 status
Device   Filename       Component                Version                 State      Progress Result
-------- -------------- ------------------------ ------------------------------- -------- ------------------
node001                 CPLDMB_0                 0.2.1.0                 current    N/A
node001                 CPLDMID_0                0.2.1.0                 current    N/A
node001                 EROT_BIOS_0              00.04.0020.0000_n00     current    N/A
node001                 EROT_BMC_0               00.04.0020.0000_n00     current    N/A
node001                 HostBIOS_0               01.00.00                current    N/A
node001                 HostBMC_0                23.00.00                current    N/A
node001                 PCIeRetimer_0            1.30.0                  current    N/A
node001                 PCIeRetimer_1            1.30.0                  current    N/A
node001                 PCIeSwitch_0             0.0.1                   current    N/A
node001                 PCIeSwitch_1             1.0.1                   current    N/A
node001                 PSU_0                    0202.0200.0200          current    N/A
node001                 PSU_1                    0202.0200.0200          current    N/A
node001                 PSU_2                    0202.0200.0200          current    N/A
node001                 PSU_3                    0202.0201.0202          current    N/A
node001                 PSU_4                    0202.0201.0203          current    N/A
node001                 PSU_5                    0202.0200.0200          current    N/A
node001  nvfw_...fwpkg  HGX_FW_BMC_0            HGX-22.10-1-rc44        completed  N/A      success: activated
node001  nvfw_...fwpkg  HGX_FW_ERoT_BMC_0       00.02.0134.0000_n00     completed  N/A      success: activated
node001  nvfw_...fwpkg  HGX_FW_ERoT_FPGA_0      00.02.0134.0000_n00     completed  N/A      success: activated
node001  nvfw_...fwpkg  HGX_FW_ERoT_NVSwitch_0  00.02.0134.0000_n00     completed  N/A      success: activated
node001  nvfw_...fwpkg  HGX_FW_ERoT_NVSwitch_1  00.02.0134.0000_n00     completed  N/A      success: activated
node001  nvfw_...fwpkg  HGX_FW_ERoT_NVSwitch_2  00.02.0134.0000_n00     completed  N/A      success: activated
```

```
node001  nvfw_...fwpkg  HGX_FW_ERoT_NVSwitch_3    00.02.0134.0000_n00   completed  N/A      success: activated
node001  nvfw_...fwpkg  HGX_FW_ERoT_PCIeSwitch_0  00.02.0134.0000_n00   completed  N/A      success: activated
node001  nvfw_...fwpkg  HGX_FW_FPGA_0             2.2C                  completed  N/A      success: activated
node001  nvfw_...fwpkg  HGX_FW_GPU_SXM_1          96.00.74.00.01        completed  N/A      success: activated
node001  nvfw_...fwpkg  HGX_FW_GPU_SXM_2          96.00.74.00.01        completed  N/A      success: activated
node001  nvfw_...fwpkg  HGX_FW_GPU_SXM_3          96.00.74.00.01        completed  N/A      success: activated
node001  nvfw_...fwpkg  HGX_FW_GPU_SXM_4          96.00.74.00.01        completed  N/A      success: activated
node001  nvfw_...fwpkg  HGX_FW_GPU_SXM_5          96.00.74.00.01        completed  N/A      success: activated
node001  nvfw_...fwpkg  HGX_FW_GPU_SXM_6          96.00.74.00.01        completed  N/A      success: activated
node001  nvfw_...fwpkg  HGX_FW_GPU_SXM_7          96.00.74.00.01        completed  N/A      success: activated
node001  nvfw_...fwpkg  HGX_FW_GPU_SXM_8          96.00.74.00.01        completed  N/A      success: activated
node001  nvfw_...fwpkg  HGX_FW_NVSwitch_0         96.10.3F.00.01        completed  N/A      success: activated
node001  nvfw_...fwpkg  HGX_FW_NVSwitch_1         96.10.3F.00.01        completed  N/A      success: activated
node001  nvfw_...fwpkg  HGX_FW_NVSwitch_2         96.10.3F.00.01        completed  N/A      success: activated
node001  nvfw_...fwpkg  HGX_FW_NVSwitch_3         96.10.3F.00.01        completed  N/A      success: activated
node001  nvfw_...fwpkg  HGX_FW_PCIeRetimer_0      2.7.9                 completed  N/A      success: activated
node001  nvfw_...fwpkg  HGX_FW_PCIeRetimer_1      2.7.9                 completed  N/A      success: activated
node001  nvfw_...fwpkg  HGX_FW_PCIeRetimer_2      2.7.9                 completed  N/A      success: activated
node001  nvfw_...fwpkg  HGX_FW_PCIeRetimer_3      2.7.9                 completed  N/A      success: activated
node001  nvfw_...fwpkg  HGX_FW_PCIeRetimer_4      2.7.9                 completed  N/A      success: activated
node001  nvfw_...fwpkg  HGX_FW_PCIeRetimer_5      2.7.9                 completed  N/A      success: activated
node001  nvfw_...fwpkg  HGX_FW_PCIeRetimer_6      2.7.9                 completed  N/A      success: activated
node001  nvfw_...fwpkg  HGX_FW_PCIeRetimer_7      2.7.9                 completed  N/A      success: activated
node001  nvfw_...fwpkg  HGX_FW_PCIeSwitch_0       1.7.5F                completed  N/A      success: activated
```

A DGX GB200 and GB300 firmware update: Procedures to upgrade DGX firmware components on the DGX GB200 and DGX GB300 hardware can be carried out after the hardware components (the compute trays, NVLink switch trays, and power shelves) are seen as being up in BCM. The procedure generally follows the firmware upgrade outlined for DGX H100 hardware (page 756). More details specific to the GB200 and GB300 hardware can be found at:

`https://docs.nvidia.com/mission-control/docs/rack-bring-up-install/2.0.0/rack-firmware-update.html#`.

The `cm-nvfwupd` module is used by `cmsh` in `firmware` command operations. BCM point release version 11.34.0 introduces module version 2.1.2, which adds support for the GB200 NVL72 rack and Rubin-generation platforms.

Setting `firmwaremanagemode` for the DGX GB200 and GB300: The `gb200` and `gb300` values (page 762) apply to the compute tray, while `gb200sw` and `gb300sw` apply to the NVLink switch tray in the NVL72 rack.

Obtaining and placing the firmware packages for the DGX GB200 and GB300: Firmware for the compute tray is available as two `.fwpkg` packages: one for the DGX-side components (BMC, EROT, CPLD) and one for the HGX-side components (BMC, CPU, GPU, EROT, FPGA, CPLD). Firmware for the NVLink switch tray is available as three `.fwpkg` packages, covering BMC/EROT/FPGA, SBIOS/EROT, and CPLD respectively.

The packages should be placed in the head node directory at

`/cm/local/apps/cmd/etc/htdocs/bios/firmware/`

under the `gb200` subdirectory for compute-tray packages, and the `gb200sw` subdirectory for switch-tray packages (the same two subdirectories are also used for GB300, since GB300 uses the same package-naming and directory scheme):

Example

```
[basecm11 ~]# ls -l /cm/local/apps/cmd/etc/htdocs/bios/firmware/gb200
-rw-r--r-- 1 root root  67349914 Nov 10 11:26 nvfw_DGX-GBX00_0023_250901.1.0_custom_prod-signed.fwpkg
-rw-r--r-- 1 root root 119706314 Nov 10 11:26 nvfw_HGX-GBX00_0023_250908.1.0_custom_prod-signed.fwpkg
[basecm11 ~]# ls -l /cm/local/apps/cmd/etc/htdocs/bios/firmware/gb200sw
-rw-r--r-- 1 root root  78998965 Nov 10 11:27 nvfw_GB200-P4978_0004_250821.1.0_prod-signed.fwpkg
-rw-r--r-- 1 root root  17018227 Nov 10 11:27 nvfw_GB200-P4978_0006_250710.1.1_prod-signed.fwpkg
-rw-r--r-- 1 root root   1729367 Nov 10 11:27 nvfw_GB200-P4978_0007_250831.1.0_prod-signed.fwpkg
```

Configuring access for the NVLink switch tray: In addition to setting `firmwaremanagemode` to `gb200sw` (or `gb300sw`), the NVLink switch tray needs a further access-settings configuration step, since its firmware commands are carried out via the NVOS REST API rather than the Redfish protocol used by the compute tray. This is configured in the `accesssettings` submode (the same submode used to configure switch access elsewhere in this manual, section 3.9.1), by setting a username and password:

Example

```
[basecm11->device[switch001]]% accesssettings
[basecm11->device[switch001]->accesssettings]% set username admin
[basecm11->device*[switch001*]->accesssettings*]% set password < password>
[basecm11->device*[switch001*]->accesssettings*]% commit
```

Managing, installing and updating the firmware package in BCM for the DGX GB200 and GB300: BCM can display information about the uploaded firmware files with the `firmware info` command, for both a compute tray device and a switch tray device:

Example

```
[basecm11->device[node001]]% firmware info
Device   Filename                        Component      Version                          State
-------- ------------------------------- -------------- -------------------------------- --------
node001  nvfw_DGX-GBX00_...fwpkg         GB200-BMC      DGX-GBX00_0023_250901.1.0_cus...  available
node001  nvfw_HGX-GBX00_...fwpkg         GB200-Compute  HGX-GBX00_0023_250908.1.0_cus...  available
[basecm11->device[switch001]]% firmware info
Device     Filename                      Component      Version                          State
---------- ----------------------------- -------------- -------------------------------- --------
switch001  nvfw_GB200-P4978_0004_...     GB200-Switch   GB200-P4978_0004_250821.1.0       available
switch001  nvfw_GB200-P4978_0006_...     GB200-Switch   GB200-P4978_0006_250710.1.1       available
switch001  nvfw_GB200-P4978_0007_...     GB200-Switch   GB200-P4978_0007_250831.1.0       available
```

The `firmware status` command displays the current firmware versions for the compute tray:

Example

```
[basecm11->device]% firmware status -n node001
Device   Filename Component               Version                    State    Progress
-------- -------- ----------------------- -------------------------- -------- --------
node001           CX8_0                   40.46.3048                 current  N/A
node001           CX8_1                   40.46.3048                 current  N/A
node001           CX8_2                   40.46.3048                 current  N/A
node001           CX8_3                   40.46.3048                 current  N/A
node001           FW_BMC_0                GB200Nvl-25.06-A            current  N/A
node001           FW_CPLD_0               0x00 0x0b 0x04 0x02         current  N/A
node001           FW_CPLD_1               0x00 0x0b 0x04 0x02         current  N/A
node001           FW_ERoT_BMC_0           01.04.0031.0000_n04         current  N/A
node001           HGX_FW_BMC_0            GB200Nvl-25.06-A            current  N/A
node001           HGX_FW_CPLD_0           0.22                        current  N/A
```

```
node001          HGX_FW_CPU_0            02.04.12                  current  N/A
node001          HGX_FW_CPU_1            02.04.12                  current  N/A
node001          HGX_FW_ERoT_BMC_0       01.04.0031.0000_n04       current  N/A
node001          HGX_FW_ERoT_CPU_0       01.04.0031.0000_n04       current  N/A
node001          HGX_FW_ERoT_CPU_1       01.04.0031.0000_n04       current  N/A
node001          HGX_FW_ERoT_FPGA_0      01.04.0008.0000_n04       current  N/A
node001          HGX_FW_ERoT_FPGA_1      01.04.0008.0000_n04       current  N/A
node001          HGX_FW_FPGA_0           1.20                      current  N/A
node001          HGX_FW_FPGA_1           1.20                      current  N/A
node001          HGX_FW_GPU_0            97.00.B9.00.68            current  N/A
node001          HGX_FW_GPU_1            97.00.B9.00.68            current  N/A
node001          HGX_FW_GPU_2            97.00.B9.00.68            current  N/A
node001          HGX_FW_GPU_3            97.00.B9.00.68            current  N/A
node001          HGX_InfoROM_GPU_0       G548.0201.00.06           current  N/A
node001          HGX_InfoROM_GPU_1       G548.0201.00.06           current  N/A
node001          HGX_InfoROM_GPU_2       G548.0201.00.06           current  N/A
node001          HGX_InfoROM_GPU_3       G548.0201.00.06           current  N/A
node001          HGX_PCIeSwitchConfig_0  01151024                  current  N/A
node001          NIC_1                   32.43.2408                current  N/A
node001          UEFI                    buildbrain-gcid-41193377  current  N/A
```

For the GB300, the output is nearly identical; the only difference is that `NIC_1` in the preceding table is reported as `NIC_0` on the GB300 compute tray.

The NVLink switch tray reports its own, shorter set of components:

Example

```
[basecm11->device]% firmware status -n switch001
Device     Filename Component    Version              State    Progress
---------- -------- ------------ -------------------- -------- --------
switch001           ASIC         35.2014.1724          current  N/A
switch001           BIOS         0ACTV_00.01.017       current  N/A
switch001           BMC          88.0002.1040          current  N/A
switch001           CPLD1        CPLD000370_REV0300    current  N/A
switch001           CPLD2        CPLD000377_REV0500    current  N/A
switch001           CPLD3        CPLD000373_REV0205    current  N/A
switch001           CPLD4        CPLD000390_REV0200    current  N/A
switch001           EROT         01.04.0026.0000_n04   current  N/A
switch001           EROT-ASIC1   01.04.0026.0000_n04   current  N/A
switch001           EROT-ASIC2   01.04.0026.0000_n04   current  N/A
switch001           EROT-BMC     01.04.0026.0000_n04   current  N/A
switch001           EROT-CPU     01.04.0026.0000_n04   current  N/A
switch001           EROT-FPGA    01.04.0026.0000_n04   current  N/A
switch001           FPGA         0.1A                  current  N/A
switch001           SSD          CE00A400              current  N/A
switch001           transceiver  N/A                   current  N/A
```

The NVLink switch tray hardware, and so this output, is the same for GB200 and GB300 racks.

Components are updated the same way as for the DGX H100, using the `firmware flash` command. Each firmware package must be flashed with its own `firmware flash` invocation: the compute tray has two packages to flash (`DGX-GBX00` for the BMC, EROT, and CPLD; then `HGX-GBX00` for the HGX components), and the NVLink switch tray has three (`GB200-P4978_0004` for the BMC, EROT, and FPGA; `GB200-P4978_0006` for the SBIOS and EROT; then `GB200-P4978_0007` for the CPLD). Flashing only some of the packages leaves the tray partially updated.

Example

```
[basecm11->device[node001]]% firmware flash nvfw_DGX-GBX00_0023_250901.1.0_custom_prod-signed.fwpkg
Device           flashing                                                 Result   Error
---------------- -------------------------------------------------------- -------- ----------
node001          nvfw_DGX-GBX00_0023_250901.1.0_custom_prod-signed.fwpkg good

[basecm11->device[node001]]% firmware flash nvfw_HGX-GBX00_0023_250908.1.0_custom_prod-signed.fwpkg
Device           flashing                                                 Result   Error
---------------- -------------------------------------------------------- -------- ----------
node001          nvfw_HGX-GBX00_0023_250908.1.0_custom_prod-signed.fwpkg good
```

The NVLink switch tray is flashed the same way, once per package:

Example

```
[basecm11->device[switch001]]% firmware flash nvfw_GB200-P4978_0004_250821.1.0_prod-signed.fwpkg
Device           flashing                                                 Result   Error
---------------- -------------------------------------------------------- -------- ----------
switch001        nvfw_GB200-P4978_0004_250821.1.0_prod-signed.fwpkg       good

[basecm11->device[switch001]]% firmware flash nvfw_GB200-P4978_0006_250710.1.1_prod-signed.fwpkg
Device           flashing                                                 Result   Error
---------------- -------------------------------------------------------- -------- ----------
switch001        nvfw_GB200-P4978_0006_250710.1.1_prod-signed.fwpkg       good

[basecm11->device[switch001]]% firmware flash nvfw_GB200-P4978_0007_250831.1.0_prod-signed.fwpkg
Device           flashing                                                 Result   Error
---------------- -------------------------------------------------------- -------- ----------
switch001        nvfw_GB200-P4978_0007_250831.1.0_prod-signed.fwpkg       good
```

Activation after flashing follows the same AC power cycle procedure as for the DGX H100 (page 761).

Managing iLO Firmware Via Base View: Firmware for HPE iLO nodes can also be managed from Base View, using the same underlying firmware upload and flash operations as the `cmsh firmware` command described earlier in this section. There is no dedicated navigation item for firmware management. Instead, `Manage Firmware` is offered as a bulk action from any device grid in Base View (for example `Devices`, or the device list within a category, node group, or configuration overlay), as well as from the `Categories`, `Node Groups`, and `Configuration Overlays` list pages themselves. On those list pages, more than one category, node group, or configuration overlay can be selected at once. Base View then expands the selection to every compute node the selected items contain, and manages firmware across that whole set in a single pass. This is typically a more convenient way to flash firmware across many nodes at once than selecting them individually from a device grid.

Selecting one or more nodes and clicking `Manage Firmware` opens a dialog with the following steps:

1. **Select Firmware Repository:** a grid lists the firmware package files available on the head node. More than one package can be selected at once, one file per hardware component. Click `Upload` to continue.
2. **Firmware Upload Summary:** a totals panel (Total, Unchanged, Pending, Finished, Failed) and a grid of the selected nodes and components are shown, with their current and upload firmware versions and status. Once the upload has finished, click `Flash` to continue.
3. **Firmware Flash Summary:** a grid shows live flashing progress for each node and component as a percentage, until each component reaches its final state. The dialog cannot be closed until flashing completes.

No Dry Run step is offered in this dialog for iLO.

Managing GB200, GB300 And DGX Firmware Via Base View: Firmware for GB200, GB300, and DGX nodes can also be managed from Base View, using the same `Manage Firmware` bulk action, and the same underlying firmware flash operations as the `cmsh firmware` command described earlier in this section. Flashing fails without an active NVIDIA Mission Control license, and the error returned states this.

Selecting one or more nodes and clicking `Manage Firmware` opens the same **Select Firmware Repository** screen used for iLO, but only one firmware package can be selected at a time here, matching the one-component-at-a-time flashing the `cmsh firmware` command also requires for these platforms. Two buttons are offered instead of `Upload`:

- **Dry Run:** pretends to carry out the flash, and opens a **Dry Run** screen. This screen shows a grid of the affected nodes and components, with their current version, the package version, and whether each is already up to date. Rows that are not up to date are highlighted. A **Force Install** checkbox is offered here, equivalent to the `cmsh firmware` command's `--force` option and needed to force a downgrade or a re-flash. Checking it requires confirming a warning first. From this screen, click `Flash` to continue.
- **Flash:** can be clicked directly from the **Select Firmware Repository** screen, skipping Dry Run, or from the Dry Run screen once its results have been reviewed. Either way, flashing itself is tracked to completion on the same **Firmware Flash Summary** screen used for iLO.

For 11.34.0, Dry Run does not work correctly on these platforms; this is fixed in 11.35.0. Until then, Flash should be used directly rather than Dry Run.

Base View does not currently offer an equivalent of the `cmsh firmware` command's `remove` operation for iLO, or a way to narrow a flash to named component targets the way the `cmsh firmware` command's `--targets` option does for these platforms; both operations are only available via `cmsh`.

DGX B200 and B300 firmware updates: Procedures to upgrade DGX firmware components on the DGX B200 and B300 hardware can be carried out once the hardware is recognized in BCM. The procedure generally follows the firmware upgrade outlined for DGX H100 hardware (page 756). More details specific to the hardware can be found at:

- `https://docs.nvidia.com/dgx/dgxb200-fw-update-guide/about.html`
- `https://docs.nvidia.com/dgx/dgxb300-fw-update-guide/index.html`

14.6 Hardware Match Check With The `hardware-profile` Data Producer

Often a large number of identical nodes may be added to a cluster. In such a case it is a good practice to check that the hardware matches what is expected. This can be done easily as follows:

1. The new nodes, say node129 to node255, are committed to a newly-created category `newbunch` as follows (output truncated):

```
[root@basecm11 ~]# cmsh -c "category add newbunch; commit"
[root@basecm11 ~]# for i in {129..255}
> do
> cmsh -c "device; set node00$i category newbunch; commit"
> done
Successfully committed 1 Devices
Successfully committed 1 Devices
```

   The preceding loop is easy to construct, and works, but it is quite slow for larger clusters, due to the time wasted in opening up `cmsh` and carrying out a `commit` command during each iteration of the for loop.

   For larger clusters the offending `for` loop can be replaced with a more elegant, but slightly trickier:

```
(echo device;
 for i in {129..255}; do
 echo "set node00$i category newbunch"
 done
 echo "commit") | cmsh
```

2. The hardware profile of one of the new nodes, say node129, is saved into the category `newbunch`. This is done using the `node-hardware-profile` health check script:

   Example

   ```
   [root@basecm11 ~]# /cm/local/apps/cmd/scripts/healthchecks/node-hardware-profile -n node129 -s newbunch
   ```

   The profile is intended to be the reference hardware against which all the other nodes should match, and is saved under the directory `/cm/shared/apps/cmd/hardware-profiles/`, and further under the directory name specified by the `-s` option, which in this case is `newbunch`.

3. The `hardware-profile` data producer (section 10.2.10) can then be enabled, and the sampling frequency set as follows:

   ```
   [root@basecm11 ~]# cmsh
   [basecm11]% monitoring setup use hardware-profile
   [basecm11->monitoring->setup[hardware-profile]]% set interval 600; set disabled no; commit
   ```

   The `hardware-profile` data producer should also be set to the category `newbunch` created in the earlier step. This can be done by creating a category group within the `nodeexecutionfilters` submode. Within that group, categories can be set for where the hardware check is to run. For the example, it is just run on one category, `newbunch`:

   ```
   [basecm11->monitoring->setup[hardware-profile]]% nodeexecutionfilters
   [basecm11->...-profile]->nodeexecutionfilters]% add category filterhwp
   [basecm11->...-profile]->nodeexecutionfilters*[filterhwp*]]% set categories newbunch
   [basecm11->...-profile]->nodeexecutionfilters*[filterhwp*]]% commit
   ```

4. CMDaemon then automatically alerts the administrator if one of the nodes does not match the hardware of that category during the first automated check. In the unlikely case that the reference node is itself faulty, then that will also be obvious because all—or almost all, if more nodes are faulty—of the other nodes in that category will then be reported "faulty" during the first check.

14.7 Serial Over LAN Console Access

Direct console access to nodes is not always possible. Other possibilities to access the node are:

1. **SSH access via an ssh client.** This requires that an `ssh` server runs on the node and that it is accessible via the network. Access can be via one of the following options:

   - a regular SSH client, run from a `bash` shell
   - via an `ssh` command run from the `device` mode of `cmsh`
   - via an `ssh` terminal launched from Base View via the navigation path:
     `Devices` > `Nodes` > *node* > `Connect` > `ssh`.

2. **Remote shell via CMDaemon.** This is possible if CMDaemon is running on the node and accessible via Base View or `cmsh`.

- In Base View, An interactive root shell session can be started up on a node via the navigation path:

  `Devices` > `Nodes` > *node* > `Connect` > `Root shell`.

  This session is connected to the node via CMDaemon, and runs `bash` by default.
- For `cmsh`, in `device` mode, running the command `rshell node001` launches an interactive bash session connected to node001 via CMDaemon.

3. **Connecting via a serial over LAN console.** If a serial console is configured, then a serial over LAN (SOL) console can be accessed from `cmsh` (`rconsole`).

Item 3 in the preceding list, SOL access, is a useful low-level access method that is covered next more thoroughly with:

- some background notes on serial over LAN console access (section 14.7.1)
- the configuration of SOL with Base View (section 14.7.2)
- the configuration of SOL with `cmsh` (section 14.7.3)
- the `conman` SOL logger and viewer (section 14.7.4)

14.7.1 Background Notes On Serial Console And SOL

Serial ports are data ports that can usually be enabled or disabled for nodes in the BIOS.

If the serial port of a node is enabled, it can be configured in the node kernel to redirect a console to the port. The serial port can thus provide what is called serial console access. That is, the console can be viewed using a terminal software such as minicom (in Linux) or Hyperterminal (in Windows) on another machine to communicate with the node via the serial port, using a null-modem serial cable. This has traditionally been used by system administrators when remote access is otherwise disabled, for example if ssh access is not possible, or if the TCP/IP network parameters are not set up right.

While traditional serial port console access as just described can be useful, it is inconvenient, because of having to set arcane serial connection parameters, use the relatively slow serial port and use a special serial cable. Serial Over LAN (SOL) is a more recent development of serial port console access, which uses well-known TCP/IP networking over a faster Ethernet port, and uses a standard Ethernet cable. SOL is thus generally more convenient than traditional serial port console access. The serial port DB-9 or DB-25 connector and its associated 16550 UART chip rarely exist on modern servers that support SOL, but they are nonetheless usually implied to exist in the BIOS, and can be "enabled" or "disabled" there, thus enabling or disabling SOL.

SOL is a feature of the BMC (Baseboard Management Controller) for IPMI 2.0 and iLO. For DRAC, CIMC, and Redfish, SOL via IPMI is used. SOL is enabled by configuring the BMC BIOS. When enabled, data that is going to the BMC serial port is sent to the BMC LAN port. SOL clients can then process the LAN data to display the console. As far as the node kernel is concerned, the serial port is still just behaving like a serial port, so no change needs to be made in kernel configuration in doing whatever is traditionally done to configure serial connectivity. However, the console is now accessible to the administrator using the SOL client on the LAN.

SOL thus allows SOL clients on the LAN to access the Linux serial console if

1. SOL is enabled and configured in the BMC BIOS
2. the serial console is enabled and configured in the node kernel
3. the serial port is enabled and configured in the node BIOS

The BMC BIOS, node kernel, and node BIOS therefore all need to be configured to implement SOL console access.

Background Notes: BMC BIOS Configuration

The BMC BIOS SOL values are usually enabled and configured as a submenu or pop-up menu of the node BIOS. These settings must be manually made to match the values in BCM, or vice versa.

During a factory reset of the node, it is likely that a SOL configuration in BCM will no longer match the configuration on the node BIOS after the node boots. This is because BCM cannot configure these. This is in contrast to the IP address and user authentication settings of the BMC (section 3.7), which BCM is able to configure on reboot.

Background Notes: Node Kernel Configuration

Sections 14.7.2 and 14.7.3 explain how SOL access configuration is set up for the node kernel using Base View or `cmsh`. SOL access configuration on the node kernel is serial access configuration on the node kernel as far as the system administrator is concerned; the only difference is that the word "serial" is replaced by "SOL" in BCM's Base View and `cmsh` front ends to give a cluster perspective on the configuration.

Background Notes: Node BIOS Configuration

Since BIOS implementations vary, and serial port access is linked with SOL access in various ways by the BIOS designers, it is not possible to give short and precise details on how to enable and configure them. The following rules-of-thumb, if followed carefully, should allow most BMCs to be configured for SOL access with BCM:

- Serial access, or remote access via serial ports, should be enabled in the BIOS, if such a setting exists.
- The node BIOS serial port settings should match the node configuration SOL settings (section 14.7.3). That means, items such as "`SOL speed`", "`SOL Flow Control`", and "`SOL port`" in the node configuration must match the equivalent in the node BIOS. Reasonable values are:
  - SOL speed: 115200bps. Higher speeds are sometimes possible, but are more likely to have problems.
  - SOL flow control: On. It is however unlikely to cause problems if flow control is off in both.
  - SOL port: COM1 (in the BIOS serial port configuration), corresponding to ttyS0 (in `x86_64`) or ttyAMA0 (in `aarch64`) in the node kernel serial port configuration. Alternatively, COM2, corresponding to ttyS1 or ttyAMA1. Sometimes, the BIOS configuration display indicates SOL options with options such as: "`COM1 as SOL`", in which case such an option should be selected for SOL connectivity.
  - Terminal type: VT100 or ANSI.
- If there is an option for BIOS console redirection after BIOS POST, it should be disabled.
- If there is an option for BIOS console redirection before or during BIOS POST, it should be enabled.
- The administrator should be aware that the BMC LAN traffic, which includes SOL traffic, can typically run over a dedicated NIC or over a shared NIC. The choice of dedicated or shared is toggled, either in the BIOS, or via a physical toggle, or both. If BMC LAN traffic is configured to run on the shared NIC, then just connecting a SOL client with an Ethernet cable to the dedicated BMC NIC port shows no console.
- The node BIOS values should manually be made to match the values in BCM, or vice versa.

14.7.2 SOL Console Configuration With Base View

In Base View, SOL configuration settings can be carried out per image via the navigation path `Provisioning` > `Software Images` > *image* > `Edit` > `Settings`

If the Enable SOL option is set to Yes then the kernel option to make the Linux serial console accessible is used after the node is rebooted.

This means that if the serial port and SOL are enabled for the node hardware, then after the node reboots the Linux serial console is accessible over the LAN via an SOL client.

If SOL is correctly configured in the BIOS and in the image, then access to the Linux serial console is possible via the minicom serial client running on the computer (from a bash shell for example), or via the rconsole serial client running in cmsh.

14.7.3 SOL Console Configuration And Access With cmsh

In cmsh, the serial console kernel option for a software image can be enabled within the softwareimage mode of cmsh. For the default image of default-image, this can be done as follows:

Example

```
[root@basecm11 ~]# cmsh
[basecm11]% softwareimage use default-image

[basecm11->softwareimage[default-image]]% set enablesol yes
[basecm11->softwareimage*[default-image*]]% commit
```

The SOL settings for a particular image can be seen with the show command:

```
[basecm11->softwareimage[default-image]]% show | grep SOL
Parameter                      Value
------------------------------ --------------
Enable SOL                     yes
SOL Flow Control               yes
SOL Port                       ttyS1
SOL Speed                      115200
```

Values can be adjusted if needed with the set command. The value for SOL Port is normally ttyS0 for x86_64 images, and ttyAMA0 for arch64 images.

On rebooting the node, the new values are used.

To access a node via an SOL client, the node can be specified from within the device mode of cmsh, and the rconsole command run on cmsh on the head node:

Example

```
[root@basecm11 ~]# cmsh
[basecm11]% device use node001
[basecm11->device[node001]]% rconsole
 screen cleared and the following conman output is displayed:
================================================================================
conman
To exit IPMI SOL, type <ENTER> "&" "."
================================================================================

<ConMan> Connection to console [node001] opened.
```

If at this point, there is no further response in conman on the console after pressing the <ENTER> key, then there is a communication failure, probably due to a misconfigured communication parameter. This could happen, for example, if the serial port ttyS1 has been set, but the node is connected on ttyS0. Setting the value of SOL Port to ttyS0 and rebooting the node to pick up the new value, would solve that issue, so that pressing the <ENTER> key, would display the node console:

Example

```
Ubuntu 18.04.2 LTS node001 ttyS0

node001 login:
```

14.7.4 The conman Serial Console Logger And Viewer

In BCM, the console viewer and logger service conman is used to connect to an SOL console and log the console output.

If the "Enable SOL" option in Base View, or if the enablesol parameter in cmsh is enabled for the software image, then the conman configuration is written out and the conman service is started.

Logging The Serial Console

The data seen at the serial console is then logged via SOL to the head node after reboot. For each node that has logging enabled, a log file is kept on the head node. For example, for node001 the log file would be at /var/log/conman/node001.log. To view the logged console output without destroying terminal settings, using less with the -R option is recommended, as in: less -R /var/log/conman/node001.log.

Using The Serial Console Interactively

Viewing quirk during boot: In contrast to the logs, the console viewer shows the initial booting stages of the node as it happens. There is however a quirk the system administrator should be aware of:

Normally the display on the physical console is a copy of the remote console. However, during boot, after the remote console has started up and been displaying the physical console for a while, the physical console display freezes. For the Linux 2.6 kernel series, the freeze occurs just before the ramdisk is run, and means that the display of the output of the launching init.d services is not seen on the physical console (figure 14.2).

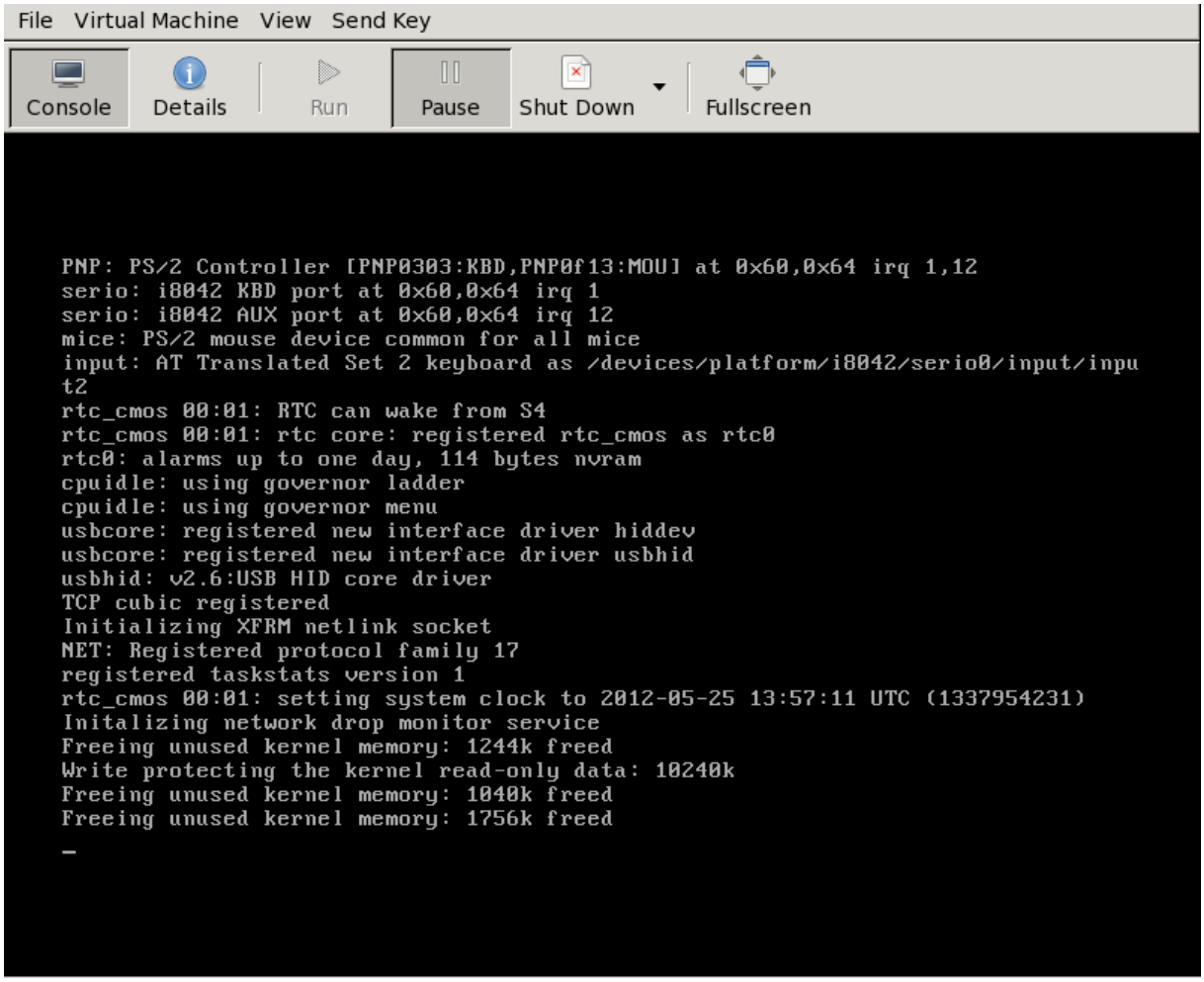


Figure 14.2: Physical Console Freeze During SOL Access

The freeze is only a freeze of the display, and should not be mistaken for a system freeze. It occurs because the kernel is configured during that stage to send to only one console, and that console is the remote console. The remote console continues to display its progress (figure 14.3) during the freeze of the physical console display.

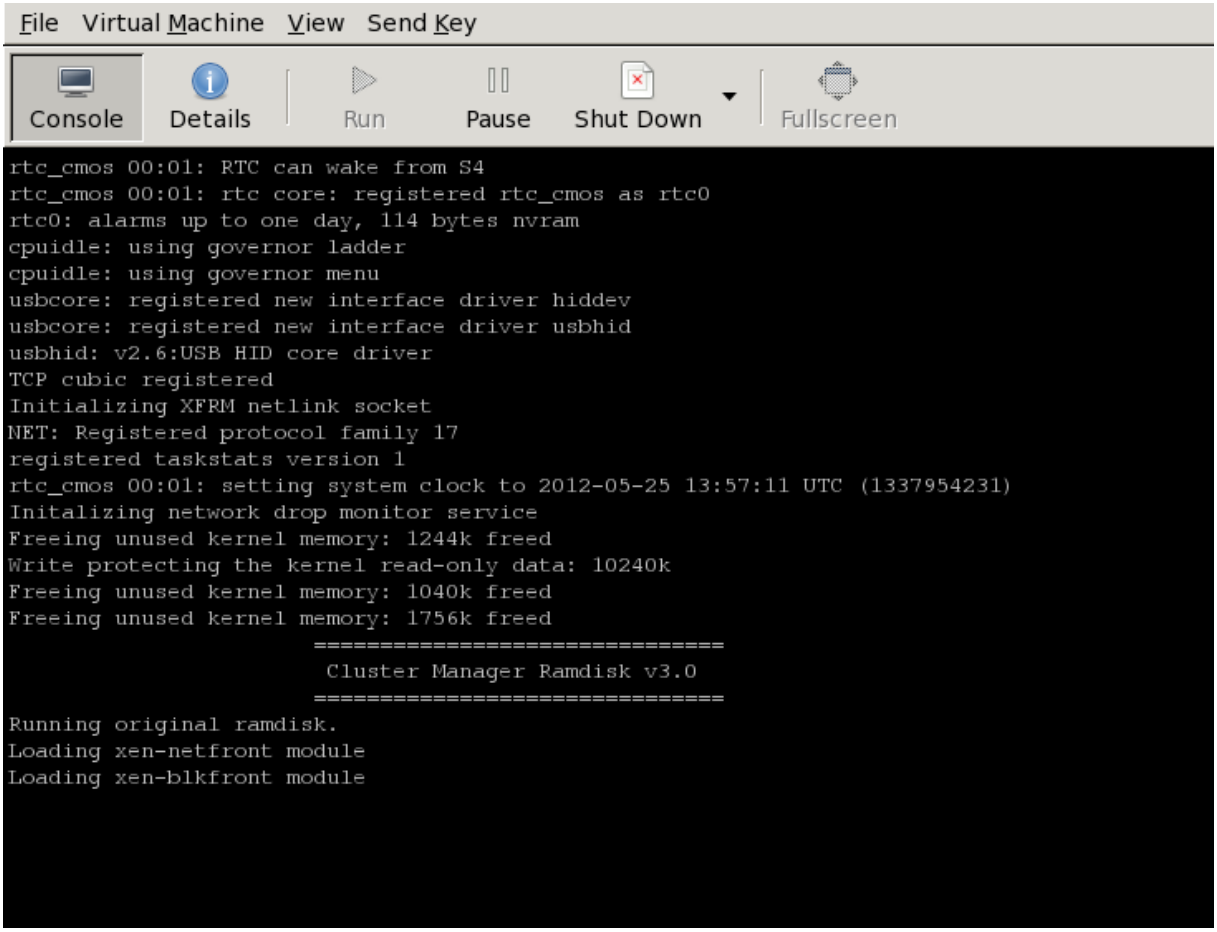


Figure 14.3: Remote Console Continues During SOL Access During Physical Console Freeze

Finally, just before login is displayed, the physical console once more (figure 14.4) starts to display what is on the remote console (figure 14.5).

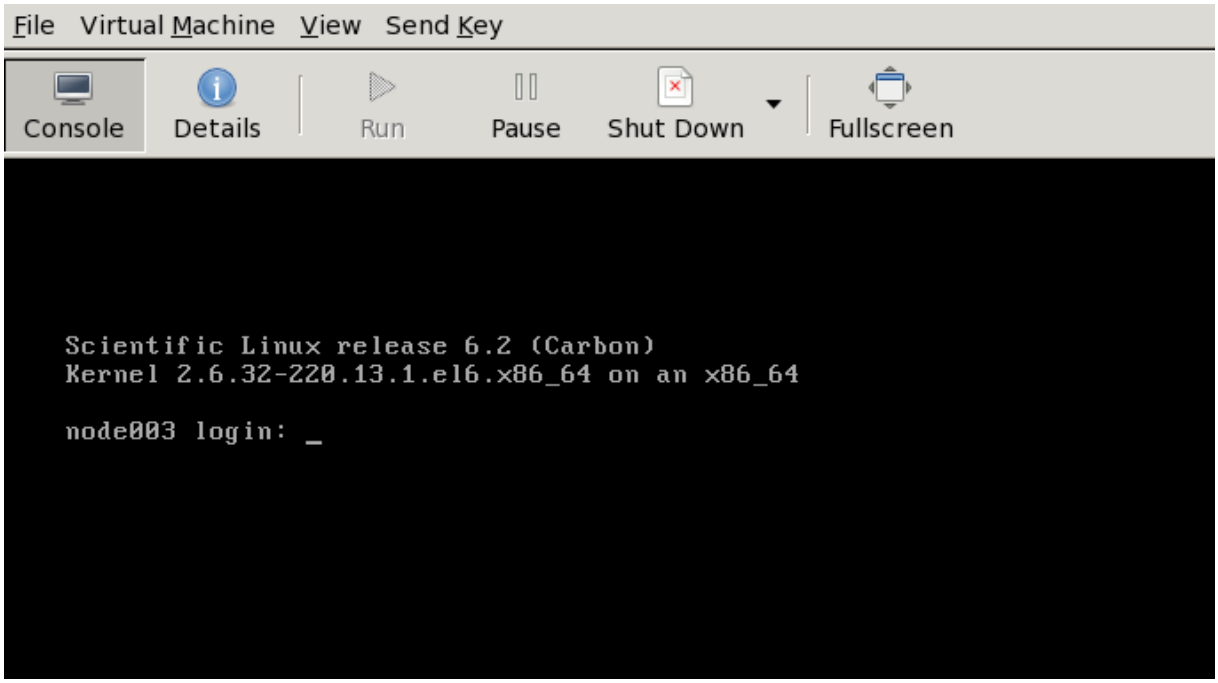


Figure 14.4: Physical Console Resumes After Freeze During SOL Access

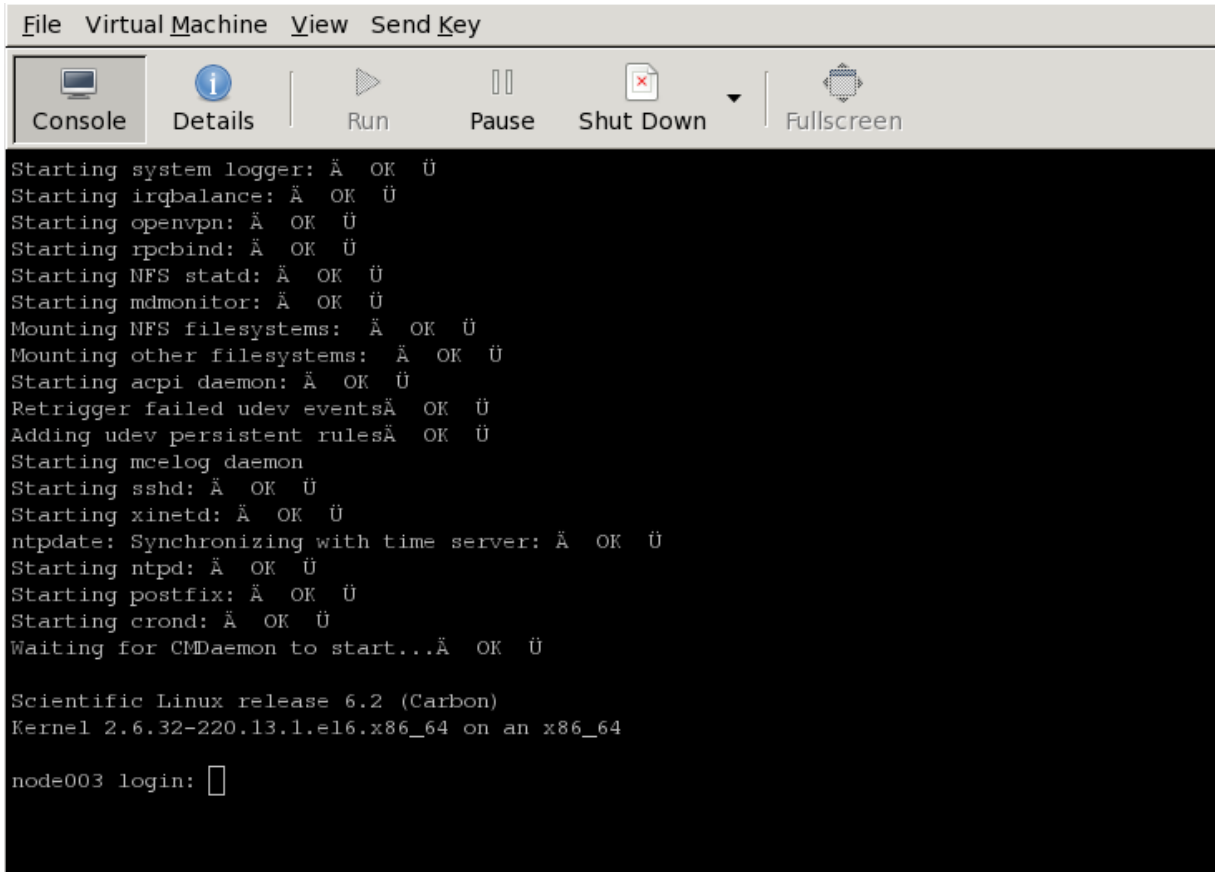


Figure 14.5: Remote Console End Display After Boot

The physical console thus misses displaying several parts of the boot progress.

Exit sequence: The `conman` console viewer session can be exited with the sequence `&.` (the last entry in the sequence being a period). Strictly speaking, the `&.` sequence must actually be preceded by an <*ENTER*>.

The console buffer issue when accessing the remote console: A feature of SOL console clients is that the administrator is not presented with any text prompt from the node that is being accessed. This is useful in some cases, and can be a problem in others.

An example of the issue is the case where the administrator has already logged into the console and typed in a command in the console shell, but has no intention of pressing the <*ENTER*> key until some other tasks are first carried out. If the connection breaks at this point, then the command typed in is held in the console shell command buffer, but is not displayed when a remote serial connection is re-established to the console—the previously entered text is invisible to the client making the connection. A subsequent <*ENTER*> would then attempt to execute the command. This is why an <*ENTER*> is not sent as the last key sequence during automated SOL access, and it is left to the administrator to enter the appropriate key strokes.

To avoid commands in the console shell buffer inadvertently being run when taking over the console remotely, the administrator can start the session with a <*CTRL*>`-u` to clear out text in the shell before pressing <*ENTER*>.

14.8 Managing Raw Monitoring Data

From NVIDIA Base Command Manager version 8.0 onward, the raw monitoring data values are stored as binary data under `/var/spool/cmd/monitoring` instead of as binary data within MySQL or MariaDB. The reason behind this change was to significantly increase performance. The monitoring subsystem in BCM was thoroughly rewritten for this change.

14.8.1 Monitoring Subsystem Disk Usage With The `monitoringinfo --storage` Option

The disk usage by the monitoring subsystem can be viewed using the `monitoringinfo` command with the `--storage` option:

Example

```
[basecm11->device]% monitoringinfo master --storage
Storage                     Elements   Disk size    Usage    Free disk
--------------------------- ---------- ------------ -------- ------------
Mon::Storage::Engine        1,523      1.00 GiB     1.28%    14.1 GiB
Mon::Storage::Message       1          16.0 MiB     0.000%   -
Mon::Storage::RepositoryId  1,528      47.7 KiB     100.0%   -
```

The Engine component stores the raw monitoring data. It grows in 1GB increments each time its usage reaches 100%.

14.8.2 Estimating The Required Size Of The Storage Device

The final size of the monitoring directory can be estimated with the script `cm-monitoring-disk-usage.py`.

The size estimate assumes that there are no changes in configuration, such as enabling advanced metrics for jobs, or increasing the maximum number of labeled entities, or large numbers of running jobs.

The size estimate value is the maximum value it will take if the cluster runs forever. It is therefore an over-estimate in practice.

Example

```
[root@basecm11 ~]# /cm/local/apps/cmd/scripts/monitoring/cm-monitoring-disk-usage.py
Number of used entities:      10
Number of used measurables:   286
Number of measurables:        286
Number of data producers:     98
Number of consolidators:      2

Current monitoring directory: /var/spool/cmd/monitoring
Monitoring directory size:    1.024 GB
Maximal directory size:       1.409 GB
```

14.8.3 Moving Monitoring Data Elsewhere

A procedure to move monitoring data from the default `/var/spool/cmd/monitoring/` directory to a new directory is as follows:

1. A new directory in which monitoring should be saved is picked.

   The block storage device for the directory should not be a shared DAS (Direct Attached Storage, such as a locally attached drive) or a NAS (Network Attached Storage, such as NFS or Lustre which work over a network connection). That is because if there is an outage, then:

   - If such a DAS storage becomes unavailable at some time, then CMDaemon assumes that no monitoring data values exist, and creates an empty data file on the local storage. If the DAS storage comes back and is mounted again, then it hides the underlying files, which would lead to discontinuous values and related issues.
   - If such a NAS storage is used, then an outage of the NAS can make CMDaemon unresponsive as it waits for input and output. In addition, when CMDaemon starts with a NAS storage, and if the NAS is unavailable for some reason, then an inappropriate mount may happen as in the DAS storage case, leading to discontinuous values and related issues.

2. The `MonitoringPath` directive (page 927) is given the new directory as its value.
3. CMDaemon is stopped (`systemctl stop cmd`).
4. The `/var/spool/cmd/monitoring/` directory is moved to the new directory.
5. CMDaemon is restarted (`systemctl start cmd`).

14.8.4 Reducing Monitoring Data By Reducing Samples

Options to reduce the amount of monitoring data gathered include reducing the `Maximal age` and `Maximal samples` for data producers (section 10.4.1) to smaller, but still non-zero values. The reduction automatically re-initializes the monitoring data collection in the background, after 30 minutes or so, so that existing data is removed, and the values reported by the `cm-monitoring-disk-usage.py` script (section 14.8.2) then show the new storage estimates for the monitoring data.

14.8.5 Deleting All Monitoring Data

A procedure to delete all monitoring data from the default `/var/spool/cmd/monitoring/` directory is as follows:

1. The CMDaemon service on all nodes can be stopped by running the following on the active head node:
   `pdsh -g all systemctl stop cmd`

2. On both head nodes, the monitoring data is removed with:

```
rm -f /var/spool/cmd/monitoring/*
rm -f /var/spool/cmd/backup/*/var/spool/cmd/monitoring/*
```

3. On both head nodes, the associated database tables for the CMDaemon user must be cleared. These tables are linked at a low level with the files under the .../spool/cmd/monitoring/ directories, which means that the tables and the directories must both be cleared or CMDaemon is likely to crash when restarted. Clearing the tables can be done with a MySQL session run on each head node.

   The CMDaemon database user is cmdaemon by default, but the value can be checked with a grep on the cmd.conf file:

```
[root@basecm11 ~]# grep ^DBUser /cm/local/apps/cmd/etc/cmd.conf
DBUser = "cmdaemon"
```

   Similarly, the password for the cmdaemon user can be found with a grep as follows:

```
[root@basecm11 ~]# grep ^DBPass /cm/local/apps/cmd/etc/cmd.conf
DBPass = "slarti8813bartfahrt"
```

   The monitoring measurables can then be deleted by running a session on each head node as follows:

   Example

```
[root@basecm11 ~]# mysql -ucmdaemon -p
Enter password:
Welcome to the MariaDB monitor.  Commands end with ; or \g.
Your MariaDB connection id is 2909
Server version: 5.5.56-MariaDB MariaDB Server

Copyright (c) 2000, 2017, Oracle, MariaDB Corporation Ab and others.

Type 'help;' or '\h' for help. Type '\c' to clear the current input statement.

MariaDB [(none)]> use cmdaemon;
Database changed
MariaDB [cmdaemon]> truncate MonitoringMeasurables;
MariaDB [cmdaemon]> truncate MonitoringMeasurableMetrics;
MariaDB [cmdaemon]> truncate MonitoringMeasurableHealthChecks;
MariaDB [cmdaemon]> truncate MonitoringMeasurableEnums;
MariaDB [cmdaemon]> truncate EntityMeasurables;
MariaDB [cmdaemon]> truncate EnumMetricValues;
MariaDB [cmdaemon]> truncate LabeledEntities;
MariaDB [cmdaemon]> truncate JobInformation;
MariaDB [cmdaemon]> truncate WlmJobPowerUsageSettingsCpuPowerMetrics;
MariaDB [cmdaemon]> truncate WlmJobPowerUsageSettingsGpuPowerMetrics;
MariaDB [cmdaemon]> truncate WlmJobPowerUsageSettingsNodePowerMetrics;
MariaDB [cmdaemon]> truncate WlmJobPowerUsageSettingsPowerUnderAllocationMetrics;
MariaDB [cmdaemon]> exit
```

   repeat on other head node

4. On both head nodes, CMDaemon can then be restarted with:
   `systemctl start cmd`

5. On the active head node, after the command:
   `cmha status`
   shows all is `OK`, the CMDaemon service can be started on all regular nodes again. The `OK` state should be achieved in about 15 seconds.

   The CMDaemon service is started with, for example:
   `pdsh -g computenode systemctl start cmd`

14.9 Node Replacement

To replace an existing node with a new node, the node information can be updated via `cmsh`.

If the new MAC address is known, then it can set that for the node. If the MAC address is not known, then the existing entry can be cleared.

If the MAC address is not known ahead of time, then the node name for the machine should be selected when it is provisioning for the first time. The steps for a new node `node031` would be as follows:

Example

```
[root@basecm11 ~]# cmsh
[basecm11]% device use node0031
if new mac address is known, then:
[basecm11->device[node031]]% set mac <new mac address>
else if new mac address is not known:
[basecm11->device[node031]]% clear mac
the changed setting in either case must be committed:
[basecm11->device[node031]]% commit
```

If the disk is the same size as the one that is being replaced, and everything else matches up, then this should be all that needs to be done

There is more information on the node installing system in section 5.4. How to add a large number of nodes at a time efficiently is described in that section. The methods used can include the `newnodes` command of `cmsh` (page 269) and the `Nodes Identification` resource of Base View (page 273).

14.10 Ansible And NVIDIA Base Command Manager

This section describes using Ansible with BCM. Using Ansible to install NVIDIA Base Command Manager is described in section 3.4 of the *Installation Manual*.

14.10.1 An Overview Of Ansible

Ansible is a popular automated configuration management software.

The BCM administrator is expected to have some experience already with Ansible. The basic concepts are covered in the official Ansible documentation at `https://docs.ansible.com/ansible/latest/user_guide/basic_concepts.html`, and further details are accessible from that site too.

As a reminder:

- Ansible is designed to administer groups of machines from an *inventory* of machines.
- An Ansible *module* is code, usually in Python, that is executed by Ansible to carry out Ansible *tasks*, usually on a remote node. The module returns values.
- An Ansible *playbook* is a YAML file. The file declares a configuration that is to be executed ("the playbook is followed") on selected machines. The execution is usually carried out over SSH, by placing modules on the remote machine.
- Traditionally, official Ansible content was obtained as a part of milestone releases of Ansible Engine, (the Red Hat version of Ansible for the enterprise).

- Since Ansible version 2.10, the official way to distribute content is via Ansible content *collections*. Collections are composed of Ansible playbooks, modules, module utilities and plugins. The collection is a formatted set of tools used to achieve automation with Ansible.
- The official Ansible list of collections is at `https://docs.ansible.com/ansible/latest/collections/index.html#list-of-collections`. At the time of writing of this section (March 2022) there were 100 collections.
- Community-supported collections are also available, at `galaxy.ansible.com`.

Picking Up The BCM Ansible Collections

In particular, the web interface at `https://galaxy.ansible.com/brightcomputing` shows the updated list of BCM Ansible collections.

From version 9.1 onward of BCM, the BCM Ansible collection naming scheme has been changed so that the name now indicates the BCM version number. This now makes it simpler for the cluster administrator to choose the right Ansible collection.

For example, to install the latest version of the BCM Ansible collection for a NVIDIA Base Command Manager 11 cluster, the following command can now be run:

```
[root@basecm11 ~]# ansible-galaxy collection install brightcomputing.bcm110
```

14.10.2 A Simple Playbook Example

In this section, a playbook from the BCM collection is run.

Preparations

To start with, Python is loaded, and Ansible installed:

```
[root@basecm11 ~]# module load python3
[root@basecm11 ~]# pip install ansible
```

Running A Simple Playbook

The directory `/cm/local/examples/cmd/ansible` has several BCM Ansible playbook examples.

The Ansible playbook to add a user can be run. The playbook is simply:

```
[root@basecm11 ~]# cat /cm/local/examples/cmd/ansible/add-user.yaml
---
- hosts: all
  gather_facts: false
  tasks:
  collections:
- brightcomputing.bcm110
  tasks:
  - name: create test-user
    user:
      name: test-user
      password: test-user-password
      profile: readonly
```

The latest NVIDIA Base Command Manager 11-compatible version of the BCM Ansible collection is at `https://galaxy.ansible.com/brightcomputing/bcm110`. It can be installed with the `ansible-galaxy` tool from the `galaxy.ansible.com` repository directly with:

```
[root@basecm11 ~]# ansible-galaxy collection install brightcomputing.bcm110
```

Documentation For The BCM Collection

The `brightcomputing.bcm110` documentation for modules can be explored using `ansible-doc` in the usual way, using the namespace. For example, for the `user` module in the `brightcomputing.bcm110` namespace, this would be (output truncated):

```
[root@basecm11 ~]# ansible-doc brightcomputing.bcm110.user
> BRIGHTCOMPUTING.BCM110.USER
> (/root/.ansible/collections/ansible_collections/brightcomputing/bcm110/plugins/modules/user.py)

        User

ADDED IN: version 9.2.0 of brightcomputing.bcm110

  * note: This module has a corresponding action plugin.

OPTIONS (= is mandatory):

- ID
        User ID number
        [Default: (null)]
        type: str

- cloneFrom
        The id or name of the entity that the new entity will be cloned from.
         (take effect only at entity creation)
        [Default: ]
        type: str

- email
...
```

The list of modules in the `brightcomputing.bcm110` collection can be viewed with `ansible-doc -l brightcomputing.bcm110`.

Almost all the modules are available as a pair. For such a pair, one module out of the pair is to query the attributes of the entity being dealt with by the pair, while the other module is to set the attributes.

Running The Ansible Playbook

The `add-user` playbook can now be run with:

Example

```
[root@basecm11 ~]# ansible-playbook -ilocalhost, /cm/local/examples/cmd/ansible/add-user.yaml

PLAY [all]
*********************************************************************************************

TASK [create test-user]
*********************************************************************************************
changed: [localhost]

PLAY RECAP
*********************************************************************************************
localhost                  : ok=1    changed=1    unreachable=0    failed=0    skipped=0
rescued=0    ignored=0
```

The YAML code shows a user should be created after execution of the playbook. If unsure, the playbook can be run again. This should do no harm since well-formed playbooks are idempotent.

The new list of users can be verified with:

```
[root@basecm11 ~]# cmsh -c "user list"
Name (key)       ID (key)         Primary group    Secondary groups
---------------- ---------------- ---------------- ----------------
cmsupport        1000             cmsupport
test-user        1001             test-user
```

14.10.3 An Intermediate Playbook Example: Setting Up A Cluster For Demonstration Purposes

The simple playbook in the preceding section has the advantage of being a quick way for the administrator to be reasonably sure that Ansible is running as it should be.

An administrator who is intending to use Ansible is typically going to need to be more familiar with how Ansible playbooks can be used to define BCM infrastructure.

This section (14.10.3) and the next (14.10.4) aim to provide this familiarity. They should be a guide for users when they go about defining their own BCM infrastructure with Ansible, as well as a model for how to carry out Ansible tasks for BCM.

The example session in this section (section 14.10.3) is about a cluster administrator who wishes to prepare a playbook so that the default image is updated, and then have the cluster set up some new objects with default values. This is useful for testing out changes in the new objects. The idea being that the administrator has up-to-date nodes with default settings in the new objects, and which work to begin with. That makes the new objects suitable for demonstrations and for making changes to see how it affects the standard settings. It also provides the convenience of being able to refer back to the working defaults in the original objects if things go wrong with the demonstration objects.

The tasks to bring the cluster to the "demo" state are described next.

Cloning The Image

The administrator now clones the default image. The idea being that further changes can be made on the cloned image later on, with the default image instance remaining unchanged, and available for comparison.

The YAML example `clone-software-image.yaml` provided with BCM can be displayed and used to carry out the cloning as shown in the following session:

```
[root@basecm11 ~]# cat /cm/local/examples/cmd/ansible/clone-software-image.yaml
---
- hosts: all
  gather_facts: false
  tasks:
    - name: clone a software image
      brightcomputing.bcm110.software_image:
        name: cloned-image
        cloneFrom: default-image
        path: /cm/images/cloned-image

[root@basecm11 ~] ansible-playbook -i localhost, /cm/local/examples/cmd/ansible/clone-software-image.yaml
```

Cloning The Category

A clone of the `default` category, `democategory`, can be built with the `brightcomputing.bcm110.category` module:

```
[root@basecm11 ~]# cat clonedefaultcat.yaml
- hosts: all
```

```
  gather_facts: false

  tasks:
    - name: clone category
      brightcomputing.bcm110.category:
        name: democategory
        cloneFrom: default
[root@basecm11 ~] ansible-playbook -i localhost, clonedefaultcat.yaml
```

Setting The Software Image In The Cloned Category To Be The Cloned Image

The software image in the cloned category can then be set to the `cloned-image` from earlier with:

```
[root@basecm11 ~]# cat setimageincat.yaml
- hosts: all
  gather_facts: false

  tasks:
    - name: set image in category
      brightcomputing.bcm110.category:
        name: democategory
        softwareImageProxy:
          parentSoftwareImage: cloned-image
[root@basecm11 ~] ansible-playbook -i localhost, setimageincat.yaml
```

Setting The Regular Nodes To Be In The Cloned Category

The regular nodes `node001` and `node002` can be placed in the cloned category with:

```
[root@basecm11 ~]# cat setcatofnodes.yaml
- hosts: all
  gather_facts: false

  tasks:
    - name: list all nodes
      brightcomputing.bcm110.node_info:
        format: dict
        include_id: false
        for_update: true
      register: result

    - name: set head_node
      set_fact:
        all_nodes: "{{ result.nodes }}"

    - name: assign compute nodes to cloned category
      brightcomputing.bcm110.physical_node:
        hostname: "{{ item }}"
        mac: "{{ all_nodes[item].mac }}"
        category: democategory
      loop:
        - node001
        - node002
[root@basecm11 ~] ansible-playbook -i localhost, setcatofnodes.yaml
```

Without using Ansible, and using `cmsh` directly instead, the preceding placement could be carried out with:

```
[root@basecm11 ~] for i in {001..002}
do cmsh -c "device use node$i; set category democategory; commit"
done
```

14.10.4 A More Complicated Playbook Example: Creating An Edge Site And Related Properties

This section provides a more complicated BCM Ansible collection-based playbook, and elaborates upon how it is used.

The collection is first shown as a whole in the following section. Then less obvious portions from it are explained, with the help of number labels, in the sections after that, starting on page 784.

The Collection

The full collection is as follows:

```
---
- hosts: all
  gather_facts: false
  vars:
    site:
      name: test-site
      secret: SECRET

    director:
      hostname: test-site-director
      mac: 00:11:22:33:44:55
      eth0_ip: 10.152.0.254
      eth1_ip: 10.161.0.254

    nodes:
      - hostname: edge-node-01
        mac: 00:11:22:33:44:01
        eth0_ip: 10.161.0.1

      - hostname: edge-node-02
        mac: 00:11:22:33:44:02
        eth0_ip: 10.161.0.2

      - hostname: edge-node-03
        mac: 00:11:22:33:44:03
        eth0_ip: 10.161.0.3

  pre_tasks:

    - name: set compute nodes for site
      set_fact:
        site_compute_nodes: "{{nodes | map(attribute='hostname') | list}}"

    - name: set nodes for site
      set_fact:
        site_nodes: "{{[director.hostname] + site_compute_nodes}}"

  tasks:
    # Network creation
    - name: create an external network
      brightcomputing.bcm110.network:
```

```
      state: present
      name: test-site-external_network
      type: EDGE_EXTERNAL
      baseAddress: 10.152.0.0
      broadcastAddress: 10.152.255.255
      netmaskBits: 16
      domainName: test-site-external_network
      management: true

  - name: create an internal network
    brightcomputing.bcm110.network:
        state: present
        type: EDGE_INTERNAL
        name: test-site-internal_network
        baseAddress: 10.161.0.0
        broadcastAddress: 10.161.255.255
        dynamicRangeStart: 10.161.16.0
        dynamicRangeEnd: 10.161.19.255
        netmaskBits: 16
        domainName: test-site-internal_network
        management: true
        bootable: true

  - name: create edge site software image
    brightcomputing.bcm110.software_image:
      name: my-software-image
      path: /cm/images/my-software-image
      cloneFrom: default-image

  - name: create edge director category
    brightcomputing.bcm110.category:
      name: edge_director_category
      softwareImageProxy:
        parentSoftwareImage: my-software-image
      fsmounts:
        - device: $localnfsserver:/cm/shared
          mountpoint:  /cm/shared
          filesystem: nfs
        - device: $localnfsserver:/home
          mountpoint:  /home
          filesystem: nfs
      state: present

  # Edge Director
  - name: create director physical node
    brightcomputing.bcm110.physical_node:
      state: present
      hostname: "{{director.hostname}}"
      partition: base
      interfaces_NetworkPhysicalInterface:
        - name: eth0
          ip: "{{ director.eth0_ip }}"
          network: test-site-external_network
        - name: eth1
          ip: "{{ director.eth1_ip }}"
```

```
          network: test-site-internal_network

      category: edge_director_category
      mac: "{{director.mac}}"
      managementNetwork: test-site-external_network
      provisioningInterface: eth0
      installBootRecord: true
      roles_EdgeDirectorRole:
        - name: edge_director_role
          openTCPPortsOnHeadNode: [636]
          externallyVisibleIp: 0.0.0.0
          externallyVisibleHeadNodeIp: 0.0.0.0
      roles_BootRole:
        - name: boot_role
          allowRamdiskCreation: true
      roles_StorageRole:
        - name: storage_role
      roles_ProvisioningRole:
        - name: provisioning_role
          allImages: LOCALDISK

  # Edge Nodes
  - name: create edge nodes
    brightcomputing.bcm110.physical_node:
      hostname: "{{item.hostname}}"
      softwareImageProxy:
        parentSoftwareImage: default-image
      interfaces_NetworkPhysicalInterface:
        - name: eth0
          ip: "{{item.eth0_ip}}"
          network: test-site-internal_network
      category: default
      mac: "{{item.mac}}"
      managementNetwork: test-site-internal_network
      installBootRecord: false
      provisioningInterface: eth0
      partition: base
    loop: "{{ nodes }}"

  - name: add test edge site
    brightcomputing.bcm110.edge_site:
      name: "{{site.name}}"
      secret: "{{site.secret}}"
      address: Springfield
      adminEmail: admin-west@email.com
      city: San Francisco
      contact: Admin
      country: USA
      notes: Note about the site
      state: present
      nodes: "{{site_nodes}}"
```

Topmost Part

```
- hosts: all #(1)
  gather_facts: false #(2)
```

1. This is a standard Ansible playbook configuration item. It defines the group or host that the playbook is run on.
2. Fact gathering is skipped here, because there is no need to use any facts that Ansible usually gathers pre-playbook-run. As a bonus, skipping it makes execution faster.

Pre_tasks Part

The `pre_tasks` section could have been made a part of the tasks section. However, making it a separate section has the benefit of separating real action from simple fact definition:

```
- name: set compute nodes for site
  set_fact:
    site_compute_nodes: "{{nodes | map(attribute='hostname') | list}}" #(1)

- name: set nodes for site
  set_fact:
    site_nodes: "{{[director.hostname] + site_compute_nodes}}" #(2)
```

1. In the preceding code, the Ansible templating capability is used to get the list of `compute_nodes` for the site that is to be created. The hostname attribute is extracted from every element of the `nodes` variable and then transformed into a list, and assigned to the `site_compute_nodes` variable.
2. The list that is created has all the nodes that are part of the site, which means the compute nodes as well as the director.

The Tasks Part

External and internal networks configuration: The tasks section starts with networking definitions (tagged here with (1) and (2)). These are the external and internal networks that are needed for the edge site that is to be created.

```
- name: create an external network    #(1)
  brightcomputing.bcm110.network:
    state: present
    name: test-site-external_network
    type: EDGE_EXTERNAL
    baseAddress: 10.152.0.0
    broadcastAddress: 10.152.255.255
    netmaskBits: 16
    domainName: test-site-external_network
    management: true

- name: create an internal network     #(2)
  brightcomputing.bcm110.network:
      state: present
      type: EDGE_INTERNAL
      name: test-site-internal_network
      baseAddress: 10.161.0.0
      broadcastAddress: 10.161.255.255
      dynamicRangeStart: 10.161.16.0
      dynamicRangeEnd: 10.161.19.255
      netmaskBits: 16
      domainName: test-site-internal_network
      management: true
      bootable: true
```

The possible values for each entity attribute can be seen by running the `ansible-doc` command:

Example

```
$ ansible-doc brightcomputing.bcm110.network
```

Director creation: A category must exist for the edge director, or must be created before the director can be created. The following snippet takes care of that:

```
- name: create edge site software image
  brightcomputing.bcm110.software_image:
    name: my-software-image
    path: /cm/images/my-software-image
    cloneFrom: default-image   #(1)

- name: create edge director category
  brightcomputing.bcm110.category:
    name: edge_director_category
    softwareImageProxy:
      parentSoftwareImage: my-software-image    #(2)
    fsmounts:     #(3)
      - device: $localnfsserver:/cm/shared
        mountpoint:  /cm/shared
        filesystem: nfs
      - device: $localnfsserver:/home
        mountpoint:  /home
        filesystem: nfs
    state: present    #(4)
```

1. The `cloneFrom` attribute is used to create the new software image from the existing one, to avoid copying over all the values that are part of the original image. The `default-image` is used here, since it is guaranteed to be defined in a new cluster.

   The `cloneFrom` attribute only takes effect when the resource is not defined. This means that if the software image is already present, then using `cloneFrom` has no effect. Removing the image allows it to be re-created again using the cloneFrom attribute.

2. The declared software image (here it is `my-software-image`) is then used to define the director category.

3. The edge directory category attributes in the snippet are standard values that are normally assigned to a director category that is to be used by director nodes.

4. The default value for `state` is `present`, so the task has the same behavior if the `state` field is left out.

Values for the director node: The edge director node values can now be set

```
- name: create director physical node
  brightcomputing.bcm110.physical_node:
    state: present
    hostname: "{{director.hostname}}"
    partition: base
    interfaces_NetworkPhysicalInterface:
      - name: eth0
        ip: "{{ director.eth0_ip }}"
        network: test-site-external_network
      - name: eth1
        ip: "{{ director.eth1_ip }}"
        network: test-site-internal_network

    category: edge_director_category
    mac: "{{director.mac}}"
```

```
    managementNetwork: test-site-external_network
    provisioningInterface: eth0
    installBootRecord: true
    roles_EdgeDirectorRole:  # (1)
      - name: edge_director_role
        openTCPPortsOnHeadNode: [636]
        externallyVisibleIp: 0.0.0.0
        externallyVisibleHeadNodeIp: 0.0.0.0
    roles_BootRole:
      - name: boot_role
        allowRamdiskCreation: true
    roles_StorageRole:
      - name: storage_role
    roles_ProvisioningRole:
      - name: provisioning_role
        allImages: LOCALDISK
```

In the preceding snippet, values are set for the physical node so that it functions correctly as a director on an edge site.

1. The director, just like the edge nodes, is just a physical node, with the role `EdgeDirectorRole` assigned to it, along with other relevant roles.

The edge (compute) nodes definition:

1. In the following snippet, the looping mechanism defines a physical node that corresponds to each declared compute node.

Each edge node belongs to the correct network.

```
- name: create edge nodes
  brightcomputing.bcm110.physical_node:
    hostname: "{{item.hostname}}"
    softwareImageProxy:
      parentSoftwareImage: default-image
    interfaces_NetworkPhysicalInterface:
      - name: eth0
        ip: "{{item.eth0_ip}}"
        network: test-site-internal_network
    category: default
    mac: "{{item.mac}}"
    managementNetwork: test-site-internal_network
    installBootRecord: false
    provisioningInterface: eth0
    partition: base
  loop: "{{ nodes }}"  #(1)
```

Edge site object creation: The last part of the playbook creates the edge site object.

```
- name: add test edge site
  brightcomputing.bcm110.edge_site:
    name: "{{site.name}}"
    secret: "{{site.secret}}"
    address: Springfield
    adminEmail: admin-west@email.com
    city: San Francisco
```

```
contact: Admin
country: USA
notes: Note about the site
state: present
nodes: "{{site_nodes}}"    #(1)
```

The `site_nodes` variable, defined by the `set_fact` task, is assigned to the `nodes` attribute of an `edge_site` action.

Running this playbook on a fresh cluster should be enough to create a new edge site with the declared properties, even if the nodes are not physically present.

The state of the edge site can be checked with `cmsh` queries. It should be noted that the image creation step may take a few minutes, depending on how big `default-image` is.

14.11 Querying Hardware Information

Querying device hardware information is about finding information about the hardware details of nodes and devices managed within the cluster. Understanding device hardware information is essential for inventory management, troubleshooting, and ensuring proper hardware configuration.

The `sysinfo` command (section 3.19.4) returns basic hardware information for a node, mostly from the OS, and using cached information.

Hardware information can also be queried using other commands available in `cmsh` device mode. These queries retrieve component details directly from the device management interfaces.

Section 14.11.1 describes using `hardwareinventoryinfo` to obtain a hardware inventory.

Section 14.11.2 describes using the BCM `lspci` command to query PCI device information.

Section 14.11.3 describes using the `lseeprom` command to query EEPROM interface information.

Section 14.11.4 describes using the `accelerators` command to query accelerator count information.

14.11.1 Hardware Inventory Information: `hardwareinventoryinfo`

A hardware inventory in accounting is taken to mean a list of items in stock in an office, a warehouse, a ship, or similar unit. In BCM `device` mode context, a hardware inventory is taken to mean the physical components installed in the cluster devices of a cluster. These physical components are components installed in a node or a switch. They can be CPUs, GPUs, network adapters, memory modules, as well as other hardware components.

The inventory data can be used for asset tracking, capacity planning, and verifying hardware configurations.

The `hardwareinventoryinfo` command uses Redfish protocol queries to display detailed hardware inventory information for a device. The information about hardware components includes the manufacturer, model, serial number, and part number for each component:

Example

```
[basecm11->device[node001]]% hardwareinventoryinfo
Device   Name     Manufacturer  Model                Serial number      ...  Part number
-------  -------- ------------ -------------------- -------------------...-------- --------------------
node001           NVIDIA        GB200 NVL            1870425000202      ...  699-25859-0311-200
node001  BMC      NVIDIA        GB200 NVL            1320325065530      ...  699-24764-0001-200
node001  CBC-0    Nvidia        18x1RU CBL Cartridge 1820025170551      ...  750-0591-000
node001  CBC-1    Nvidia        18x1RU CBL Cartridge 1820025170557      ...  750-0591-000
node001  CBC-2    Nvidia        18x1RU CBL Cartridge 1820025170552      ...  750-0591-000
node001  CBC-3    Nvidia        18x1RU CBL Cartridge 1820125170045      ...  750-0591-000
node001  CPLD-0   Intel         MAX10 10M08          NA                 ...  115-3953-000
node001  CPLD-0   Lattice       BP                   0x0044387556108c26 ...  999-0099-000
node001  CPLD-1   Lattice       BP                   0x0044387556107637 ...  999-0099-000
node001  CPLD-2   Lattice       IPEX                 0x0044387824104e2b ...  999-0099-000
```

```
node001  CPLD-3   Lattice       IPEX                 0x00443885661c8c3a ...          999-0099-000
node001  CPU0     NVIDIA        Grace A02P           0x0000000017830A0420...7008100  692-2G548-0001-000
node001  CPU1     NVIDIA        Grace A02P           0x0000000017830A0012...5000280  692-2G548-0001-000
node001  GPU0     NVIDIA        GB200 186GB HBM3e    1650625019617       ...         NA
node001  GPU1     NVIDIA        GB200 186GB HBM3e    1650625019617       ...         NA
node001  GPU2     NVIDIA        GB200 186GB HBM3e    1650625019220       ...         NA
node001  GPU3     NVIDIA        GB200 186GB HBM3e    1650625019220       ...         NA
node001  PDB      NVIDIA        P5859 B01            1870425000202       ...         699-25859-0311-200
```

The `hardwareinventoryinfo` command has options to run it per node, category, rack, and so on. The help text to the command (`help hardwareinventoryinfo`) provides details on these and other options.

14.11.2 PCI Device Information: `lspci`

The `lspci` command is run from `device` mode, and displays information about PCI devices connected to a device. The information includes the PCI device class, vendor, PCI device IDs, and whether the PCI device is a DPU or accelerator:

Example

```
[basecm11->device[node001]]% lspci
Hostname Class ID  Class          ... Vendor   ... Vendor ID  Device ID  Name        ... DPU  Accelerator
-------- --------- ---------------... ---------... ---------- ---------- ------------... ---- ------------
node001  0200      Ethernet contro... Intel Cor... 8086       1533       I210 Gigabit... no   no
node001  0200      Ethernet contro... Mellanox ... 15b3       a2dc       MT43244 Blue... no   no
node001  0200      Ethernet contro... Mellanox ... 15b3       a2dc       MT43244 Blue... no   no
node001  0207      Infiniband cont... Mellanox ... 15b3       1023       CX8 Family [... no   no
node001  0207      Infiniband cont... Mellanox ... 15b3       1023       CX8 Family [... no   no
node001  0207      Infiniband cont... Mellanox ... 15b3       1023       CX8 Family [... no   no
node001  0207      Infiniband cont... Mellanox ... 15b3       1023       CX8 Family [... no   no
node001  0300      VGA compatible ... ASPEED Te... 1a03       2000       ASPEED Graph... no   no
node001  0302      3D controller  ... NVIDIA Co... 10de       31c2       B300        ... no   yes
node001  0302      3D controller  ... NVIDIA Co... 10de       31c2       B300        ... no   yes
...
```

The `lspci` command has options to run it per node, category, rack, and so on. The help text to the command (`help lspci`) provides details on these and other options.

14.11.3 EEPROM Information: `lseeprom`

The `lseeprom` command is run from `device` mode, and displays EEPROM information for network interaces on a device. The information about transceiver modules includes the interface type, vendor, part number, serial number, connector type, and manufacturing date code:

Example

```
[basecm11->device[node001]]% lseeprom
Hostname  Interface       Type          Vendor  P/N             S/N             Connector    Date Code
--------- --------------- ------------ ------- -------------- -------------- ------------ ---------
node001   enP22p3s0f0np0  0x1e (QSFP+) NVIDIA  MMA1Z00-NS400  MT2526FT06847  MPO Parallel 250623
node001   enP22p3s0f1np1  0x1e (QSFP+) NVIDIA  MMA1Z00-NS400  MT2445FT09229  MPO Parallel 241105
...
```

The `lseeprom` command has options to run it per node, category, rack, and so on. The help text to the command (`help lseeprom`) provides details on these and other options.

14.11.4 Accelerator Count: accelerators

The accelerators command is run from device mode, and displays an overview of the number of accelerators, such as GPUS, on each device:

Example

```
[basecm11->device]% accelerators
Node                 Count
-------------------- -----
head1                0
head2                0
schedulernode        0
...
computenode001       4
computenode002       4
computenode003       4
...
```

The accelerators command has options to run it per node, category, rack, and so on. The help text to the command (help accelerators) provides details on these and other options.

14.12 The cm-mysql-setup Utility

14.12.1 Introduction

In order to manage MySQL databases on compute nodes, the NVIDIA Base Command Manager cm-mysql-setup utility has been introduced. It is a plugin that comes with the cm-setup package. It supports installation and configuration of MySQL on a single node, or on a primary/replica setup.

All distributions except Ubuntu 24 onward need a MySQL repository setup package to be downloaded from the MySQL site in advance to install the repositories for MySQL Community Edition. For Ubuntu, from Ubuntu 24 onward, the distribution MySQL packages are used.

The cm-mysql-setup plugin has been developed with Slurm integration (figure 7.12) in mind, but it is usable with other applications too, with some manual configuration.

All the nodes referred to in this section must be UP at the time of the command line run, otherwise commands will fail.

High availability Design

When high availability is needed, two nodes can be selected for deployment. The cm-mysql-setup utiility sets up a ReplicaSet with a single primary and a single replica. Replication between the primary and replica is asynchronous, for faster writes.

Replication uses GTID (Global Transaction ID) to achieve the best reliability possible. This setup installs mysql-router on the client nodes, to achieve transparency for the client daemon such as slurmdbd, when MySQL server replication is configured . This setup requires mysql-shell to be installed on the head node and on client nodes, when MySQL server replication is configured, and therefore mysql-router is also required.

If the chosen configuration is high availability, then in /root/mysql-ha-config.conf there is a configuration file containing all the passwords needed to set up the configuration. It is the root user's responsibility to keep that configuration file safe from non-root-users.

14.12.2 Installation Through The Wizard

Running cm-mysql-setup brings up the TUI installation wizard for MySQL. Deployment of MySQL is done by the cluster administrator with the following steps:

1. First screen: If MySQL is already installed on the cluster, then the screen offers an uninstallation option.

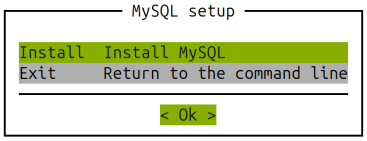


Figure 14.6: Base ViewMySQL Installation: first screen of the wizard

2. Setting a configuration overlay name for the MySQL overlay. The default value is usually fine.

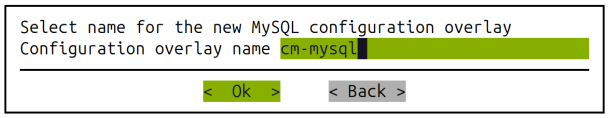


Figure 14.7: Base View MySQL Installation: MySQL overlay

3. Selecting a category in which to install MySQL. Only categories with one or two nodes can be selected, and only one category can be selected.

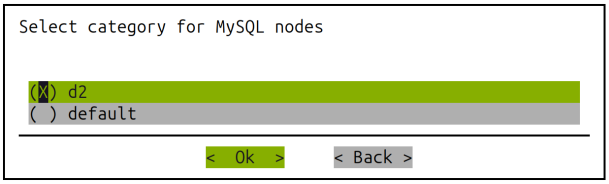


Figure 14.8: Base View MySQL Installation: Select category

4. If a category was not selected in the previous step, then one or two compute nodes can be selected now instead.

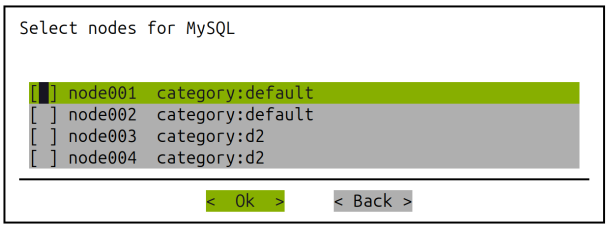


Figure 14.9: Base View MySQL Installation: Select nodes

5. If two nodes were selected in the previous step, a further screen appears to select which of the two nodes is the primary. The other node becomes the replica. Exactly one node must be selected; selecting none or both is rejected. This corresponds to the `--primary-node` option described in Section 14.12.3.

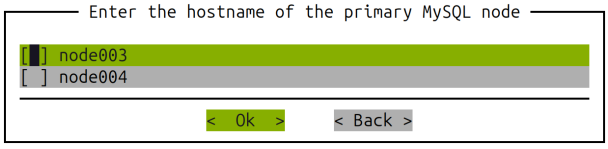


Figure 14.10: Base View MySQL Installation: Select primary node

6. On Ubuntu24, MySQL binds only to `localhost`. Here the cluster administrator can choose whether to have it bind it to all interfaces. This is what is expected if applications such as `slurmdbd` are running on other hosts.

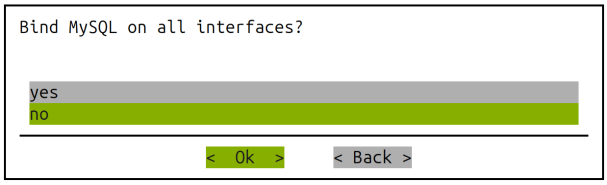


Figure 14.11: Base View MySQL Installation: Bind all interfaces

7. If this is a reinstallation with an existing data directory, the old root password is needed to configure the system, and will be changed later. If not reinstalling, the old root password is left empty.

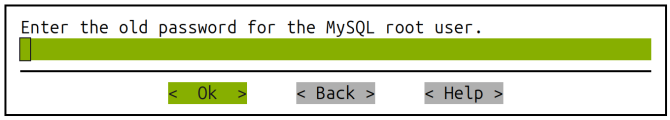


Figure 14.12: Base View MySQL Installation: give old root password

8. The new root password for MySQL can be set.

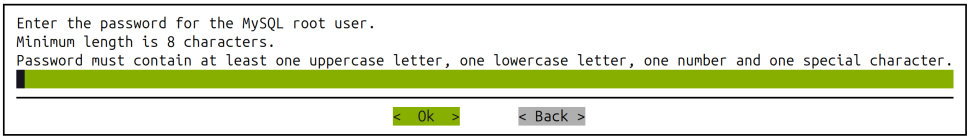


Figure 14.13: Base View MySQL Installation: set new root password

9. After these steps, the configuration can be:
   - saved and installed right away with the `Save config & deploy` option
   - saved without installing right away, and the installation carried out with that configuration later, using various other options available in the TUI

In the final summary screen, as with other scripts that come with the `cm-setup` package, the configuration setting can be one of the following kinds:

- `minimal config`: For this minimal kind of configuration setting, the configuration consists only of the settings that differ from the default application configuration. The remaining default application configuration settings are not included in the configuration.
- `config`: For this non-minimal kind of configuration setting, the configuration consists of the minimal configuration settings (the ones changed by BCM), as well as the remaining default configuration settings (the ones provided with the application). That is to say: the remaining default configuration application settings are included in the configuration. The default application configuration settings may change as the upstream packages change.

14.12.3 Installation From CLI

This command line format that is used to install `MySQL` on one or two cluster nodes is:

```
[root@basecm11 ~]# cm-mysql-setup --setup --nodes=node001,node002 --root-password='YourSecureP@ss1'
[--overlay-name=cm-mysql] [--category=db-category] [--primary-node=node001] [--skip-packages]
```

- `--nodes` and `--category` are mutually exclusive
- `--nodes` takes two existing hostnames at the most

- The category selected with --category must exist and must have either one or two nodes in it.
- The option --primary-node is compulsory if MySQL is to be installed on two nodes.
- The password in option --root-password sets the root password for the new MySQL configuration. It must be at least 8 characters long, with at least: one uppercase letter, one lowercase letter, one number, and one special character.
- --skip-packages skips the installation of the packages and the repository. The packages must always be present on the target machines.

Installation in an Air-Gapped Environment

In an air-gapped environment, the installation process is slightly different, and differs slightly for each distribution.

For Ubuntu 22.04, SLES and RHEL, the MySQL Community Edition packages are needed, while For Ubuntu 26.04 and Ubuntu 24.04, the distribution MySQL packages are used.

After that, in every case, you need to use the option --skip-packages to skip the installation of the packages and the repository package. All cm-mysql-setup commands are still valid with that addition.

It is important to note that you have to install the packages in specific software images.

- mysql-server packages are needed in server nodes' software image.
- mysql-router packages are needed in client nodes' software image, to run the helper script from there to check the routers. They are not necessary if there is a single server node selected for installation of MySQL server.
- mysql-shell packages are needed in the head nodes' software image. They are not necessary if mysql-router is not installed on the head node.
- mysql-shell packages are needed in the client nodes' software images, when mysql-router is installed on them. They are not necessary if mysql-router is not installed on the client nodes.

Before actually starting the installation with the cm-mysql-setup utility, you need to run the following commands in cmsh:

```
[root@basecm11 ~]# cmsh
[basecm11->root]% device
[basecm11->device]% foreach -n node001..node004 (imageupdate -w)
[basecm11->device]% exit
```

In the example, node001..node004 is a list of all nodes involved in the installation, which should be replaced with the actual list of nodes needing the packages to be installed.

Ubuntu 24.04 and Ubuntu 26.04 For Ubuntu 24.04 and Ubuntu 26.04, the distribution MySQL packages are used and of course they must be downloaded in advance, along with their dependencies.

You need to run the following commands to download the packages, in an online system:

```
[root@basecm11 ~]# apt-get download \
$(apt-cache depends --recurse -i mysql-server | grep '^\w' | sort -u)
[root@basecm11 ~]# apt-get download \
$(apt-cache depends --recurse -i mysql-router | grep '^\w' | sort -u)
[root@basecm11 ~]# apt-get download \
$(apt-cache depends --recurse -i mysql-shell | grep '^\w' | sort -u)
```

After that you need to move them to the software image and installing the packages in a chroot environment.

```
[root@basecm11 ~]# mv *.deb /path/to/software/image/
[root@basecm11 ~]# cm-chroot-sw-img /path/to/software/image/
[root@basecm11 software/image]% dpkg -i *.deb
[root@basecm11 software/image]% exit
```

There are a few caveats:

- If you are getting errors about conflicting packages with mariadb-connector-c or other client packages, do not remove them, as they might be needed for `slurmdbd` package. Force the installation if necessary.
- The commands in the examples for Ubuntu 22.04 work also on Ubuntu 24.04 and Ubuntu 26.04, even though the package names are different.

Ubuntu 22.04 MySQL Server Community Edition packages are used and they must be downloaded in advance. You can download packages from `https://dev.mysql.com/downloads/repo/apt/`. Version 8.0, 8.4 and 9.7 LTS are supported.

Otherwise, you can install in an online system the package configuring the MySQL Community Edition repository `https://dev.mysql.com/get/mysql-apt-config_0.8.34-1_all.deb` as it is required in the online installation.

```
[root@basecm11 ~] # wget https://dev.mysql.com/get/mysql-apt-config_0.8.34-1_all.deb
[root@basecm11 ~] # dpkg -i mysql-apt-config_0.8.34-1_all.deb
[root@basecm11 ~] # apt-get update
[root@basecm11 ~] # apt install -d -o Dir::Cache::archives=/tmp/debs mysql-community-server
[root@basecm11 ~] # apt install -d -o Dir::Cache::archives=/tmp/debs mysql-router-community
[root@basecm11 ~] # apt install -d -o Dir::Cache::archives=/tmp/debs mysql-shell
```

The previous commands must be run in an online system. When installing the repository package, chose mysql-8.0, mysql-8.4-lts or mysql-9.7-lts. Keep Connectors enabled. The commands in example download packages in the /tmp/debs directory.

There are a few caveats:

- If you are getting errors about conflicting packages with mariadb-connector-c or other client packages, do not remove them, as they might be needed for `slurmdbd` package. Force the installation if necessary.
- Before installing the server packages, apply the package installation workarounds on Ubuntu described in paragraph 14.12.3.
- The commands in the examples for Ubuntu 24.04 and Ubuntu 26.04 work also on Ubuntu 22.04, even though the package names are different.

RHEL MySQL Server Community Edition packages are used and they must be downloaded in advance. You can download the repository package from `https://dev.mysql.com/downloads/repo/yum/`. After that you can install the package in an online system and download the packages with yum from there.

```
[root@basecm11 ~] # rpm -ivh mysql84-community-release-el9-4.noarch.rpm
[root@basecm11 ~] # yum module -y disable mysql
[root@basecm11 ~] # dnf download --resolve --destdir=/tmp/rpms mysql-community-server
[root@basecm11 ~] # dnf download --resolve --destdir=/tmp/rpms mysql-router-community
[root@basecm11 ~] # dnf download --resolve --destdir=/tmp/rpms mysql-shell
```

The previous commands must be run in an online system.

SLES MySQL Server Community Edition packages are used and they must be downloaded in advance. You can download the repository package from https://dev.mysql.com/downloads/repo/suse/. After that you can install the package in an online system and download the packages with zypper from there.

```
[root@basecm11 ~] # rpm -ivh mysql84-community-release-sl15-3.noarch.rpm
[root@basecm11 ~] # zypper --installroot /root/mysql-server \
  --pkg-cache-dir /tmp/rpms \
  in -d --no-recommends mysql-community-server
[root@basecm11 ~] # zypper --installroot /root/mysql-server \
  --pkg-cache-dir /tmp/rpms \
  in -d --no-recommends mysql-router-community
[root@basecm11 ~] # zypper --installroot /root/mysql-server \
--pkg-cache-dir /tmp/rpms \
in -d --no-recommends mysql-shell
```

In the previous examples, /root/mysql-server and /tmp/rpms should be already existing.

Ubuntu Server Package Installation Workaround Before installing the server packages, you need to:

- Create /etc/mysql/mysql.conf.d/temp_skip_networking.cnf in the image.
- Add the following content to the file:

  ```
  [mysqld]
  skip-networking=1
  ```

- Install the server packages.
- Remove the file /etc/mysql/mysql.conf.d/temp_skip_networking.cnf from the image. Failure to do so will cause the installation to fail.

14.12.4 Removal From CLI

```
[root@basecm11 ~]# cm-mysql-setup --disable --overlay-name=cm-mysql --yes-i-really-mean-it
```

--overlay-name must be an existing overlay. This command does not uninstall packages and it does not remove data from the disk. Afterwards, a new configuration can be resumed on the same nodes, with the same root password and the same data, running the same configuration. If the new configuration is different, errors might occur and manual recovery might be necessary.

14.12.5 Clean Up

```
[root@basecm11 ~]# cm-mysql-setup --cleanup --nodes=node001,node002 --yes-i-really-mean-it [--keep-helper-config]
```

The purpose of the --cleanup option is to allow MySQL to be reinstalled when the existing data is not wanted, or when the old root password is not available. As described in Section 14.12.2, the old root password is needed when a reinstallation is carried out over an existing data directory. If the data directory is removed first, then the reinstallation is started from scratch, and the old root password is no longer needed.

This command is used to remove the MySQL data from the node disks. No overlay must exist on the nodes for MySQL before running the command. This command cannot be undone and it is destructive. It should not be used on head nodes, and elsewhere should be used only when the existing data is genuinely no longer needed. The --keep-helper-config option is added if the helper configuration file is to be kept across the installations. *Important:* If the plan is to reinstall, then either all nodes should be cleaned, or all nodes should be kept unchanged. A mixed state, where some nodes contain existing data and others do not, can cause unexpected issues and potential data loss.

14.12.6 Maintenance

A helper script, `cm-slurm-mysql-ha`, is installed under `/cm/shared/apps/cm-slurm-mysql-ha/bin/` by `cm-mysql-setup` when two nodes are used for MySQL.

Its help text can be viewed with the `-h|--help` option. The usage is:

```
usage: cm-slurm-mysql-ha [-h] [--config CONFIG] [--no-save] [--generate-passwords]
                         [--generated-password-length N] [-v]
                         {setup-admin-user,setup-replicaset,setup-router,status,failover,
                         remediate,migrate-to-gtid,migrate-from-mariadb}
                         ...
```

The `cm-slurm-mysql-ha` utility uses the helper configuration file in `/root/mysql-ha-config.conf`

Failover

Manual intervention is necessary for failover, and the helper script can then be used, as in the following example:

```
cm-slurm-mysql-ha --config /root/mysql-ha-config.conf failover --host node001
```

The `--host` argument is set by the administrator to point to the node to connect to, i.e., the current primary node. It automatically selects the other MySQL node (the current replica) as the next primary.

If the primary node has crashed, then it is possible to connect to the secondary node instead, but in that case the parameter `--force` must be used.

Remediation

Remediation is the operation needed to re-synchronize a replica server (secondary) with the primary after the replica has been offline or has fallen out of sync. It removes the replica from the ReplicaSet, clones the current data from the primary onto the replica, and then rejoins the replica to the ReplicaSet. The command must be run from the head node:

```
cm-slurm-mysql-ha --config /root/mysql-ha-config.conf remediate --host primary --target secondary
```

where `--host` is the current primary node and `--target` is the replica node to be re-synchronized. Remediation is non-destructive to the primary's data, but replaces all data on the target node with a fresh clone from the primary.

Post-Failover Recovery

After a forced failover (section 14.12.6) in which the old primary has crashed and the replica has been promoted to primary, the old primary node must be re-integrated when it comes back online. At that point the old primary is out of sync and must be remediated as a replica of the new primary. The steps to be taken are:

1. Confirming that the new primary is healthy and that the old primary node is back online and reachable.

2. Running remediation, specifying the new primary as `--host` and the recovered node as `--target`:

   ```
   cm-slurm-mysql-ha --config /root/mysql-ha-config.conf remediate --host node002 --target node001
   ```

   where `node002` is the new primary (promoted during failover) and `node001` is the recovered former primary, now rejoining as replica.

3. Verifying the ReplicaSet status:

   ```
   cm-slurm-mysql-ha --config /root/mysql-ha-config.conf status
   ```

Both nodes should show as members of the ReplicaSet, with node002 as primary and node001 as replica.

The administrator may then optionally fail back to the original primary if the original node assignment is preferred, by running a normal (non-forced) failover:

```
cm-slurm-mysql-ha --config /root/mysql-ha-config.conf failover --host node002
```

Disk Failure Recovery

The recovery procedure depends on which node's disk has failed.

Disk failure on the replica (secondary):

The primary continues serving normally. When the replica's disk is repaired or the node is rebuilt remediation is run to re-sync the replica from the primary:

```
cm-slurm-mysql-ha --config /root/mysql-ha-config.conf remediate --host node001 --target node002
```

Disk failure on the primary:

This is equivalent to a primary crash. This should require the forced failover and post-failover recovery procedure:

1. The replica is promoted to primary with a forced failover (section 14.12.6):

   ```
   cm-slurm-mysql-ha --config /root/mysql-ha-config.conf failover --host node002 --force
   ```

   Any transactions that were not yet replicated to the replica at the time of the crash will be lost, since replication is asynchronous.

2. The failed primary node drive must then be repaired or rebuilt

3. MySQL is reinstalled on the repaired node.

4. The repaired node is re-integrated as a replica using remediation, as described in Post-Failover Recovery section 14.12.6.

Split Brain

A split brain situation arises when both nodes lose connectivity to each other but both remain running and potentially serving clients independently. In a two-node MySQL ReplicaSet, this can occur if the network link between the nodes is interrupted while both nodes are healthy.

Detection:

A split brain can be suspected when:

- Both nodes are reachable from the head node but cannot reach each other.
- The ReplicaSet status reports an error on one or both members.
- Applications report inconsistent query results or connection errors.

The ReplicaSet status should be checked from the head node:

```
cm-slurm-mysql-ha --config /root/mysql-ha-config.conf status
```

Resolution:

The goal is to designate one node as the authoritative primary and discard any divergent writes on the other. Because replication is asynchronous, writes that reached only one node during the split cannot be automatically reconciled, which means that manual intervention is needed. Steps in the intervention are:

1. Deciding which node's data is authoritative. Normally this is the node that was the primary before the split.

2. Stopping MySQL on the node whose data is to be discarded, to prevent further writes.
3. Restoring network connectivity between the nodes.
4. Running remediation to re-clone the authoritative node's data onto the discarded node:

```
cm-slurm-mysql-ha --config /root/mysql-ha-config.conf remediate --host node001 --target node002
```

5. Restartin MySQL on the remediated node and verifying ReplicaSet status.

Any writes that occurred on the discarded node during the split are permanently lost. If those writes must be recovered, a manual data reconciliation using MySQL binary logs is required before running remediation. The MySQL documentation can be consulted for this procedure.

Adding Custom Applications To A Compute Node

If adding a custom application to a compute node, then `mysql-router` must be set up on the node where the application is to run. `mysql-router` can be set up with:

```
cm-slurm-mysql-ha --config /root/mysql-ha-config.conf setup-router
```

The configuration file will have to be copied manually to the target node, and back, to keep the configuration synchronized between all applications.

Backup

The `cm-mysql-setup` utility does not manage backups. MySQL documentation should be consulted to find out the best practices for backups for MySQL.

15

High Availability

15.0 Introduction

15.0.1 Why Have High Availability?

In a cluster with a single head node, the head node is a single point of failure for the entire cluster. It is often unacceptable that the failure of a single machine can disrupt the daily operations of a cluster.

High availability configuration for a head node is about configuring an extra head node to provide the head node services in a redundant manner. If one head node fails, then the other head node can take over, thus providing the same services with a minimum of downtime.

High availability can be set up for other types of nodes too.

15.0.2 High Availability—For What Nodes?

By default, in this and other chapters, HA is about a head node failover configuration. When it is otherwise, then it is made explicitly clear in the manuals that it is regular node HA, or edge director HA, or COD head node HA that is being discussed.

High Availability For Head Nodes

By default, the head node usually runs the most services. The high availability (HA) feature of BCM therefore allows clusters to be set up with two head nodes configured as a failover pair, with one member of the pair being the active head. The purpose of this design is to increase availability to beyond that provided by a single head node.

High Availability For Regular Nodes

Especially with smaller clusters, it is often convenient to run all services on the head node. However, an administrator may want or need to run a service on a regular node instead. For example, a workload manager, or NFS could be run on a regular node. If a service disruption is unacceptable here, too, then HA can be configured for regular nodes too (section 15.5). HA for regular nodes is a more recent feature in BCM, and is done differently compared with head nodes.

High Availability For Edge Directors

Edge directors manage edge nodes and manage them in a similar way to how head nodes manage regular nodes. Also similar to head nodes is that edge directors can also be configured for HA (section 2.1.1 of the *Edge Manual*). However, their HA design is based on that of HA for regular nodes.

High Availability For COD Head Nodes

Cluster On Demand (COD) head nodes are head nodes that run on a cloud service provider. COD HA head nodes are very similar to standard cluster HA (on-premises) head nodes. COD HA is discussed separately in section 2.14 of the *Cloudbursting Manual*.

15.0.3 High Availability Usually Uses Shared Storage

HA is typically configured using shared storage (section 15.1.5), such as from an NFS service, which typically provides the `/home` directory on the active (section 15.1.1) head, and on the regular nodes.

15.0.4 Organization Of This Chapter

The remaining sections of this chapter are organized as follows:

- **HA On Head Nodes**
  - Section 15.1 describes the concepts behind HA, keeping the BCM configuration in mind.
  - Section 15.2 describes the normal user-interactive way in which the BCM implementation of a failover setup is configured.
  - Section 15.3 describes the implementation of the BCM failover setup in a less user-interactive way, which avoids using the ncurses dialogs of section 15.2
  - Section 15.4 describes how HA is managed with BCM after it has been set up.
- **HA On Regular Nodes**
  - Section 15.5 describes the concepts behind HA for regular nodes, and how to configure HA for them.
- **HA And Workload Manager Jobs**
  - Section 15.6 describes the support for workload manager job continuation during HA failover.

15.1 HA Concepts

15.1.1 Primary, Secondary, Active, Passive

Naming: In a cluster with an HA setup, one of the head nodes is named the *primary* head node and the other head node is named the *secondary* head node.

Mode: Under normal operation, one of the two head nodes is in *active* mode, whereas the other is in *passive* mode.

The difference between naming versus mode is illustrated by realizing that while a head node which is primary always remains primary, the mode that the node is in may change. Thus, the primary head node can be in passive mode when the secondary is in active mode. Similarly the primary head node may be in active mode while the secondary head node is in passive mode. As an aside: the definition for primary in HA for NVIDIA Base Command Manager should not be confused with the definition for primary that is used by workload managers such as Slurm when a failover mechanism is configured by the workload manager (section 7.2.4).

The difference between active and passive is that the active head takes the lead in cluster-related activity, while the passive follows it. Thus, for example, with MySQL transactions, CMDaemon carries them out with MySQL running on the active, while the passive trails the changes. This naturally means that the active corresponds to the master, and the passive to the slave, in the MySQL master-slave replication mode that MySQL is run as.

By default, only the `cmdaemon` and `slurm_acct_db` databases are cloned to the secondary head node. Other MySQL databases on the head node, such as those used by third-party applications, are not replicated (section 15.1.3).

15.1.2 Monitoring The Active Head Node, Initiating Failover

In HA the passive head node continuously monitors the active head node. If the passive finds that the active is no longer operational, it will initiate a *failover sequence*. A failover sequence involves taking over resources, services and network addresses from the active head node. The goal is to continue providing services to compute nodes, so that jobs running on these nodes keep running.

15.1.3 Services In BCM HA Setups

There are several services being offered by a head node to the cluster and its users.

Services Running On Both Head Nodes

One of the design features of the HA implementation in BCM is that whenever possible, services are offered on both the active as well as the passive head node. This allows the capacity of both machines to be used for certain tasks (e.g. provisioning), but it also means that there are fewer services to move in the event of a failover sequence.

On a default HA setup, the following key services for cluster operations are always running on both head nodes:

- **CMDaemon**: providing certain functionality on both head nodes (e.g. provisioning)
- **DHCP**: load balanced setup
- **TFTP**: requests answered on demand, under xinetd
- **LDAP**: running in replication mode (the active head node LDAP database is pulled by the passive)
- **MySQL**: running in master-slave replication mode (the active head node MySQL database is pulled by the passive). By default only the `cmdaemon` and `slurm_acct_db` databases are replicated
- **NTP**
- **DNS**
- **Workload Management**: For each of the Slurm, LSF services, one WLM is fully active on one head node, while the other head node has WLM components on a passive standby

MySQL replication scope: By default, head node MySQL HA clones and replicates only the `cmdaemon` and `slurm_acct_db` databases to the secondary. Additional named databases can be included by listing them in `extradbclone.xml`. Third-party application databases that are not listed there, such as the AJR database, are not cloned, and are therefore missing from the new active head node after a failover.

From BCM point release version 11.34.0 onwards, `cmha-setup` can optionally replicate all non-system MySQL databases and users/grants, instead of just the default set described earlier on. In the TUI, after the MySQL root password is entered, a prompt asks: "`Replicate all non-system MySQL databases?`" Choosing `Yes` removes the default database filters so that all non-system databases, along with MySQL users and grants, are cloned to the secondary and then replicated.

Non-system databases means all user-created databases, excluding MySQL's built-in schemas (`mysql`, `information_schema`, `performance_schema`, `sys`). For reference, the system and non-system databases distinguished in the preceding text are:

- **System MySQL databases** (never replicated): `mysql` (user accounts, privileges, and grants), `information_schema` (metadata about tables and columns; virtual, so not replicated in any case), `performance_schema` (runtime performance metrics), and `sys` (convenience views over `performance_schema`).
- **Non-system MySQL databases** (replicated when `Replicate all non-system MySQL databases?` is answered `Yes`): `cmdaemon`, `slurm_acct_db`, and any third-party application databases, such as the AJR database.

NOTE: In express mode the same behavior is selected with the `--replicate-all-databases` option (section 15.3).

When an HA setup is created from a single head node setup, the above services are automatically reconfigured to run in the HA environment over two head nodes.

Provisioning role runs on both head nodes In addition, both head nodes also take up the *provisioning role*, which means that nodes can be provisioned from both head nodes. As the passive head node is then also provisioned from the active, and the active can switch between primary and secondary, it means both heads are also given a value for `provisioninginterface` (section 5.4.7).

For a head node in a single-headed setup, there is no value set by default. For head nodes in an HA setup, the value of `provisioninginterface` for each head node is automatically set up by default to the interface device name over which the image can be received when the head node is passive.

The implications of running a cluster with multiple provisioning nodes are described in detail in section 5.2. One important aspect described in that section is how to make provisioning nodes aware of image changes.

From the administrator's point of view, achieving awareness of image changes for provisioning nodes in HA clusters is dealt with in the same way as for single-headed clusters. Thus, if using `cmsh`, the `updateprovisioners` command from within `softwareimage` mode is used, whereas if Base View is used, then the navigation path `Provisioning` > `Provisioning requests` > `Update provisioning nodes` can be followed (section 5.2.4).

Services That Migrate To The Active Node

Although it is possible to configure any service to migrate (become active) from one head node to another in the event of a failover, in a typical HA setup only the following services migrate:

- NFS
- The User Portal
- Workload management:
  - The Slurm DBD accounting daemon (`slurmdbd`)
  - The LSF management batch daemon (`mbatchd`), external load information managers (`elim.*`), external authentication (`eauth`), and batch scheduling manager (`mbschd`)

15.1.4 Failover Network Topology

A two-head failover network layout is illustrated in figure 15.1.

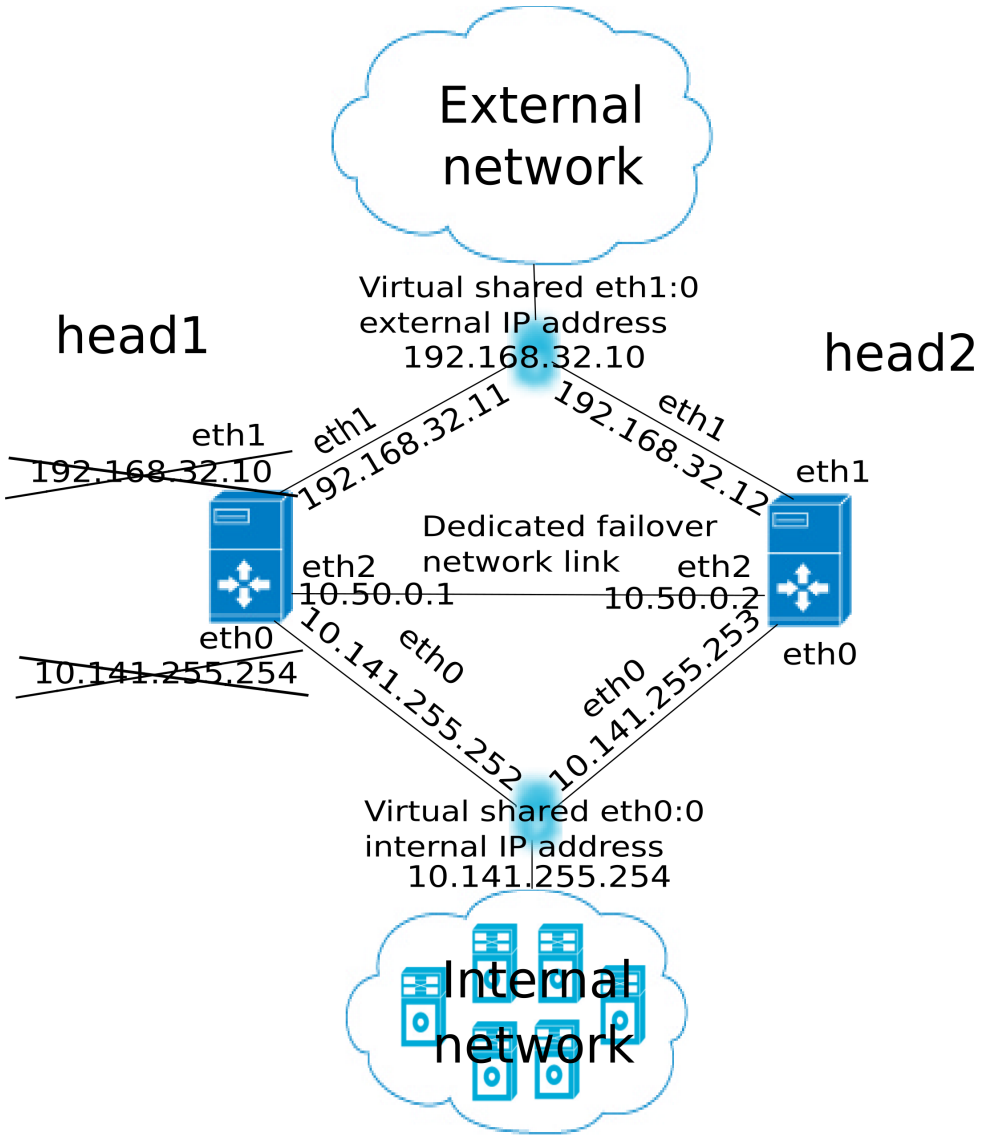


Figure 15.1: High Availability: Two-Head Failover Network Topology

In the illustration, the primary `head1` is originally a head node before the failover design is implemented. It is originally set up as part of a Type 1 network (section 3.3.9 of the *Installation Manual*), with an internal interface eth0, and an external interface eth1.

When the secondary head is connected up to help form the failover system, several changes are made.

HA: Network Interfaces

Each head node in an HA setup typically has at least an external and an internal network interface, each configured with an IP address.

In addition, an HA setup uses two virtual IP interfaces, each of which has an associated virtual IP address: the external shared IP address and the internal shared IP address. These are shared between the head nodes, but only one head node can host the address and its interface at any time.

In a normal HA setup, a shared IP address has its interface hosted on the head node that is operating in active mode. On failover, the interface migrates and is hosted on the head node that then becomes active.

When head nodes are also being used as login nodes, users outside of the cluster are encouraged

to use the shared external IP address for connecting to the cluster. This ensures that they always reach whichever head node is active. Similarly, inside the cluster, nodes use the shared internal IP address wherever possible for referring to the head node. For example, nodes mount NFS filesystems on the shared internal IP interface so that the imported filesystems continue to be accessible in the event of a failover.

Shared interfaces are implemented as alias interfaces on the physical interfaces (e.g. `eth0:0`). They are activated when a head node becomes active, and deactivated when a head node becomes passive.

HA: Dedicated Failover Network

In addition to the normal internal and external network interfaces on both head nodes, the two head nodes are usually also connected using a direct dedicated network connection, `eth2` in figure 15.1. This connection is used between the two head nodes to monitor their counterpart's availability. It is called a *heartbeat* connection because the monitoring is usually done with a regular heartbeat-like signal between the nodes such as a ping, and if the signal is not detected, it suggests a head node is dead.

To set up a failover network, it is highly recommended to simply run a UTP cable directly from the NIC of one head node to the NIC of the other, because not using a switch means there is no disruption of the connection in the event of a switch reset.

15.1.5 Shared Storage

Almost any HA setup also involves some form of shared storage between two head nodes to preserve state after a failover sequence. For example, user home directories must always be available to the cluster in the event of a failover.

In the most common HA setup, the following two directories are shared:

- `/home`, the user home directories
- `/cm/shared`, the shared tree containing applications and libraries that are made available to the nodes

The shared filesystems are only available on the active head node. For this reason, it is generally recommended that users log in via the shared IP address, rather than ever using the direct primary or secondary IP address. End-users logging into the passive head node by direct login may run into confusing behavior due to unmounted filesystems.

BCM versions 11.0 and beyond use NAS (Network Attached Storage) for shared storage. Versions prior to 11.0 also allowed DAS (Direct Attached Storage) for shared storage. However, DAS by its very nature has drawbacks that make it significantly more fragile for shared storage purposes. DAS is therefore no longer suggested as a possible shared storage option for BCM.

NAS

In a Network Attached Storage (NAS) setup, both head nodes mount a shared volume from an external network attached storage device. In the most common situation this would be an NFS server either inside or outside of the cluster. Lustre or GPFS storage are other popular choices.

Because imported mounts can typically not be re-exported (which is true at least for NFS), nodes typically mount filesystems directly from the NAS device.

Custom Shared Storage With Mount And Unmount Scripts

The cluster management daemon on the two head nodes deals with shared storage through a *mount script* and an *unmount script*. When a head node is moving to active mode, it must acquire the shared filesystems. To accomplish this, the other head node first needs to relinquish any shared filesystems that may still be mounted. After this has been done, the head node that is moving to active mode invokes the *mount script* which has been configured during the HA setup procedure. When an active head node is requested to become *passive* (e.g. because the administrator wants to take it down for maintenance without disrupting jobs), the *unmount script* is invoked to release all shared filesystems.

By customizing the *mount* and *unmount* scripts, an administrator has full control over the form of shared storage that is used. Also an administrator can control which filesystems are shared.

Mount scripts paths can be set via `cmsh` or Base View (section 15.4.6).

15.1.6 Guaranteeing One Active Head At All Times

Because of the risks involved in accessing a shared filesystem simultaneously from two head nodes, it is vital that only one head node is in active mode at any time. To guarantee that a head node that is about to switch to active mode will be the only head node in active mode, it must either receive confirmation from the other head node that it is in passive mode, or it must make sure that the other head node is powered off.

What Is A Split Brain?

When the passive head node determines that the active head node is no longer reachable, it must also take into consideration that there could be a communication disruption between the two head nodes. Because the "brains" of the cluster are communicatively "split" from each other, this is called a *split brain* situation.

Since the normal communication channel between the passive and active may not be working correctly, it is not possible to use only that channel to determine either an inactive head or a split brain with certainty. It can only be suspected.

Thus, on the one hand, it is possible that the head node has, for example, completely crashed, becoming totally inactive and thereby causing the lack of response. On the other hand, it is also possible that, for example, a switch between both head nodes is malfunctioning, and that the active head node is still up and running, looking after the cluster as usual, and that the head node in turn observes that the passive head node seems to have split away from the network.

Further supporting evidence from the dedicated failover network channel is therefore helpful. Some administrators find this supporting evidence an acceptable level of certainty, and configure the cluster to decide to automatically proceed with the failover sequence, while others may instead wish to examine the situation first before manually proceeding with the failover sequence. The implementation of automatic vs manual failover is described in section 15.1.7. In either implementation, *fencing*, described next, takes place until the formerly active node is powered off.

Going Into Fencing Mode

To deal with a suspected inactive head or split brain, a passive head node that notices that its active counterpart is no longer responding, first goes into *fencing* mode from that time onward. While a node is fencing, it will try to obtain proof via another method that its counterpart is indeed inactive.

Fencing, incidentally, does not refer to a thrust-and-parry imagery derived from fencing swordplay. Instead, it refers to the way all subsequent actions are tagged and effectively fenced-off as a backlog of actions to be carried out later. If the head nodes are able to communicate with each other before the passive decides that its counterpart is now inactive, then the fenced-off backlog is compared and synced until the head nodes are once again consistent.

Ensuring That The Unresponsive Active Is Indeed Inactive

There are two ways in which "proof" can be obtained that an unresponsive active is inactive:

1. By asking the administrator to manually confirm that the active head node is indeed powered off
2. By performing a power-off operation on the active head node, and then checking that the power is indeed off to the server. This is also referred to as a STONITH (Shoot The Other Node In The Head) procedure

It should be noted that just pulling out the power cable is not quite the same as a power-off operation (section 15.2.4), because the power-off operation only powers off the main board, and not the BMC board. Both cases (pulling out the power cable or carrying out the power-off operation) trigger a failover,

but for the case of the power cable being pulled out, the BMC board is no longer, the failover that is triggered needs to be confirmed manually.

There is also a distinction between a graceful shutdown and a power-off operation, in that a graceful shutdown does not power off the main board unless that is also specified. A failover is not triggered with a graceful shutdown because it is assumed that the shutdown is not intended to cause a failover.

Once a guarantee has been obtained that the active head node is powered off, the fencing head node (i.e. the previously passive head node) moves to active mode.

Improving The Decision To Initiate A Failover With A Quorum Process

While the preceding approach guarantees one active head, a problem remains.

In situations where the passive head node loses its connectivity to the active head node, but the active head node is communicating without a problem to the entire cluster, there is no reason to initiate a failover. It can even result in undesirable situations where the cluster is rendered unusable if, for example, a passive head node decides to power down an active head node just because the passive head node is unable to communicate with any of the outside world (except for the PDU feeding the active head node).

One technique used by BCM to reduce the chances of a passive head node powering off an active head node unnecessarily is to have the passive head node carry out a quorum procedure. All nodes in the cluster are asked by the passive node to confirm that they also cannot communicate with the active head node. If more than half of the total number of nodes confirm that they are also unable to communicate with the active head node, then the passive head node initiates the STONITH procedure and moves to active mode.

15.1.7 Automatic Vs Manual Failover

Administrators have a choice between creating an HA setup with automatic or manual failover.

- In the case of an automatic failover, an active head node is powered off when it is no longer responding at all, and a failover sequence is initiated automatically.
- In the case of a manual failover, the administrator is responsible for initiating the failover when the active head node is no longer responding. No automatic power off is done, so the administrator is asked to confirm that the previously active node is powered off.

For automatic failover to be possible, power control must be defined for both head nodes. If power control is defined for the head nodes, then automatic failover is attempted by default.

The administrator may disable automatic failover. In `cmsh` this is done by setting the `disableautomaticfailover` property, which is a part of the HA-related parameters (section 15.4.6):

```
[root@basecm11 ~]# cmsh
[basecm11]% partition failover base
[basecm11->partition[base]->failover]% set disableautomaticfailover yes
[basecm11->partition*[base*]->failover*]% commit
```

With Base View it is carried out using the navigation path `Cluster > Settings > Failover > Disable automatic failover`

If no power control has been defined, or if automatic failover has been disabled, or if the power control mechanism is not working (for example due to inappropriate, broken or missing electronics or hardware), then a failover sequence must always be initiated manually by the administrator.

Sometimes, if automatic failover is enabled, but the active head is still slightly responsive (the so-called *mostly dead* state, described in section 15.4.2), then the failover sequence must also be initiated manually by the administrator.

15.1.8 HA And Cloud Nodes

As far as the administrator is concerned, HA setup remains the same whether a Cluster Extension (Chapter 3 of the *Cloudbursting Manual*) is configured or not, and whether a Cluster On Demand (Chapter 2 of the *Cloudbursting Manual*) is configured or not. Behind the scenes, on failover, any networks associated with the cloud requirements are taken care of by BCM.

15.1.9 HA Using Virtual Head Nodes

Two physical servers are typically used for HA configurations. However, each head node can also be a virtual machine (VM). The use case for this might be to gain experience with an HA configuration.

Failover Network Considerations With HA VMs

With physical head nodes in an HA configuration, the failover network, used for HA heartbeats, is typically provided by running a network cable directly between the ethernet port on each machine. Not having even a switch in between is a best practice. Since the head nodes are typically in the same, or adjacent racks, setting this up is usually straightforward.

With VMs as head nodes in an HA configuration, however, setting up the failover network can be more complex:

- The cluster administrator may need to consider if HA is truly improved by, for example, connecting the failover network of the physical node to a switch.
- Often a virtual switch would be used between the virtual head nodes, just because it is often easier.
- Not using a failover network is also an option, just as in the physical case.
- If one head node is on one hypervisor, and another is on a second hypervisor, then a standard BCM setup cannot have one head node carry out an automated failover STONITH because it cannot contact the other hypervisor. So powering off the VM in the other hypervisor would have to be done manually. An alternative to this, if automated failover is required, is to create custom power scripts.

There are no specific guidelines for the network configuration of HA with VMs in BCM. The process of configuration is however essentially the same as for a physical node.

Size Considerations With HA VMs

A virtual head node in practice may be configured with fewer CPUs and memory than a physical head node, just because such configurations are more common options in a VM setup than for a physical setup, and cheaper to run. However, the storage requirement is the same as for a physical node. The important requirement is that the head nodes should have sufficient resources for the cluster.

15.2 HA Setup Procedure Using `cmha-setup`

After installation (Chapter 3 of the *Installation Manual*) and license activation (Chapter 4 of the *Installation Manual*) an administrator may wish to add a new head node, and convert BCM from managing an existing single-headed cluster to managing an HA cluster.

Is An HA-Enabled License Required?

To convert a single-headed cluster to an HA cluster, the existing cluster license should first be checked to see if it allows HA. The `verify-license` command run with the `info` option can reveal this in the MAC address field:

Example

```
verify-license info | grep ^MAC
```

A `cmsh`-based check can also be used (`cmsh -c "main licenseinfo"| grep ^MAC`).

HA-enabled clusters display two MAC addresses in the output. Single-headed clusters show only one.

If an HA license is not active then it should first be specified and installed (Chapter 4 of the *Installation Manual*).

Existing User Certificates Become Invalid

Installing the new license means that any existing user certificates will lose their validity (page 68 of the *Installation Manual*) on Base View session logout. This means:

- If LDAP is managed by BCM, then on logout, new user certificates are generated, and a new Base View login session picks up the new certificates automatically.
- For LDAPs other than that of BCM, the user certificates need to be regenerated.

It is therefore generally good practice to have an HA-enabled license in place before creating user certificates and profiles if there is an intention of moving from a single-headed to an HA-enabled cluster later on.

The `cmha-setup` Utility For Configuring HA

The `cmha-setup` utility is a special tool that guides the administrator in building an HA setup from a single head cluster. It is not part of the cluster manager itself, but is a cluster manager tool that interacts with the cluster management environment by using `cmsh` to create an HA setup. Although it is in theory also possible to create an HA setup manually, using either Base View or `cmsh` along with additional steps, this is not supported, and should not be attempted as it is error-prone.

A basic HA setup is created in three stages:

1. **Preparation** (section 15.2.1): the configuration parameters are set for the shared interface and for the secondary head node that is about to be installed.
2. **Cloning** (section 15.2.2): the secondary head node is installed by cloning it from the primary head node.
3. **Shared Storage Setup** (section 15.2.3): the method for shared storage is chosen and set up.

An optional extra stage is:

4. **Automated Failover Setup** (section 15.2.4): Power control to allow automated failover is set up.

15.2.1 Preparation

The following steps prepare the primary head node for the cloning of the secondary. The preparation is done only on the primary, so that the presence of the secondary is not actually needed during this stage.

0. It is recommended that all nodes except for the primary head node are powered off, in order to simplify matters. The nodes should in any case be power cycled or powered back on after the basic HA setup stages (sections 15.2.1-15.2.3, and possibly section 15.2.4) are complete.
1. If bonding (section 3.5) is to be used on the head node used in an HA setup, then it is recommended to configure and test out bonding properly before carrying out the HA setup.
2. To start the HA setup, the `cmha-setup` command is run from a `root` shell on the primary head node.
3. `Setup` is selected from the main menu (figure 15.2).
4. `Configure` is selected from the `Setup` menu.

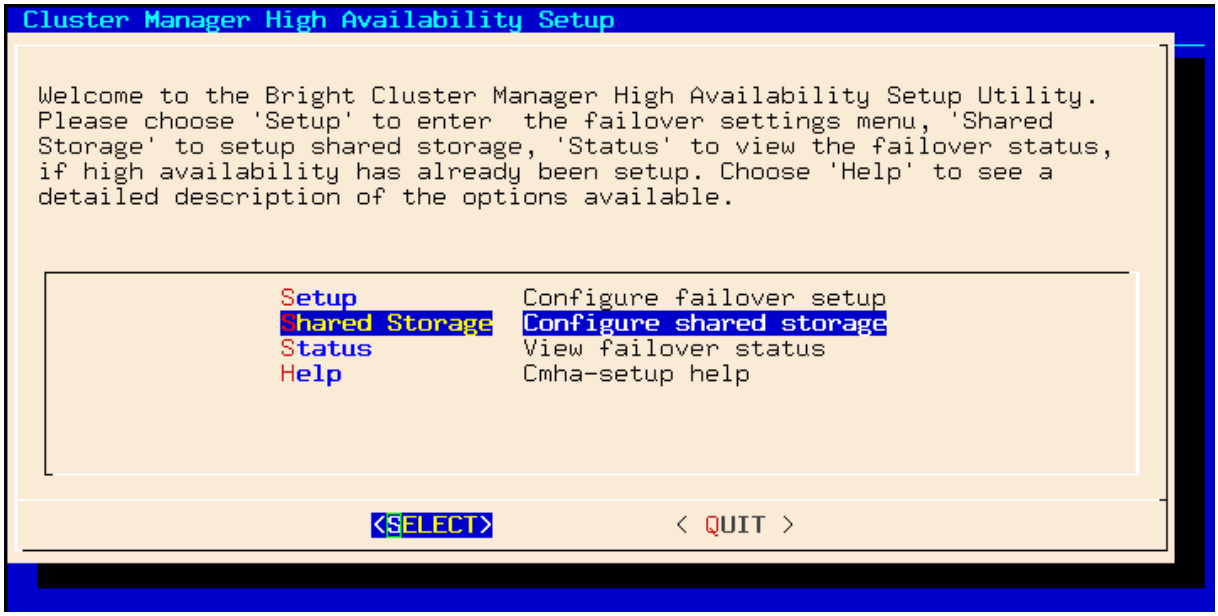


Figure 15.2: cmha-setup Main menu

5. A license check is done. Only if successful does the setup proceed further. If the cluster has no HA-enabled license, a new HA-enabled license must first be specified and activated during a request-license session (section 4.3 of the *Installation Manual*).
6. The virtual shared internal alias interface name and virtual shared internal IP alias address are set.
7. The virtual shared external alias interface name and virtual shared external IP alias address are set. For the external shared virtual IP address as well as for the external regular IP addresses, each head node external interface address must be a static IP addresses for the HA configuration. Attempting to use DHCP for external addresses in HA is not going to work.
8. The host name of the passive is set.
9. Failover network parameters are set. The failover network physical interface should exist, but the interface need not be up. The network name, its base address, its netmask, and domain name are set. This is the network used for optional heartbeat monitoring.
10. Failover network interfaces have their name and IP address set for the active and passive nodes.
11. The primary head node may have other network interfaces (e.g. InfiniBand interfaces, a BMC interface, alias interface on the BMC network). These interfaces are also created on the secondary head node, but the IP address of the interfaces still need to be configured. For each such interface, when prompted, a unique IP address for the secondary head node is configured.
12. The network interfaces of the secondary head node are reviewed and can be adjusted as required. DHCP assignments on external interfaces can be set by setting the value DHCP. If the primary head node has a DHCP-assigned IP address, then the input field for the secondary head node is set by default to the value DHCP.
13. A summary screen displays the planned failover configuration. If alterations need to be made, they can be done via the next step.
14. The administrator is prompted to set the planned failover configuration. If it is not set, the main menu of cmha-setup is re-displayed.

15. If the option to set the planned failover configuration is chosen, then a password for the MySQL root user is requested.

16. After the MySQL password is entered, the administrator is asked whether to replicate all non-system MySQL databases. The default is to clone only the `cmdaemon` and `slurm_acct_db` databases to the secondary. Choosing `Yes` replicates all non-system databases and MySQL users/grants (section 15.1.3).

17. Setup progress for the planned configuration is displayed (figure 15.3).

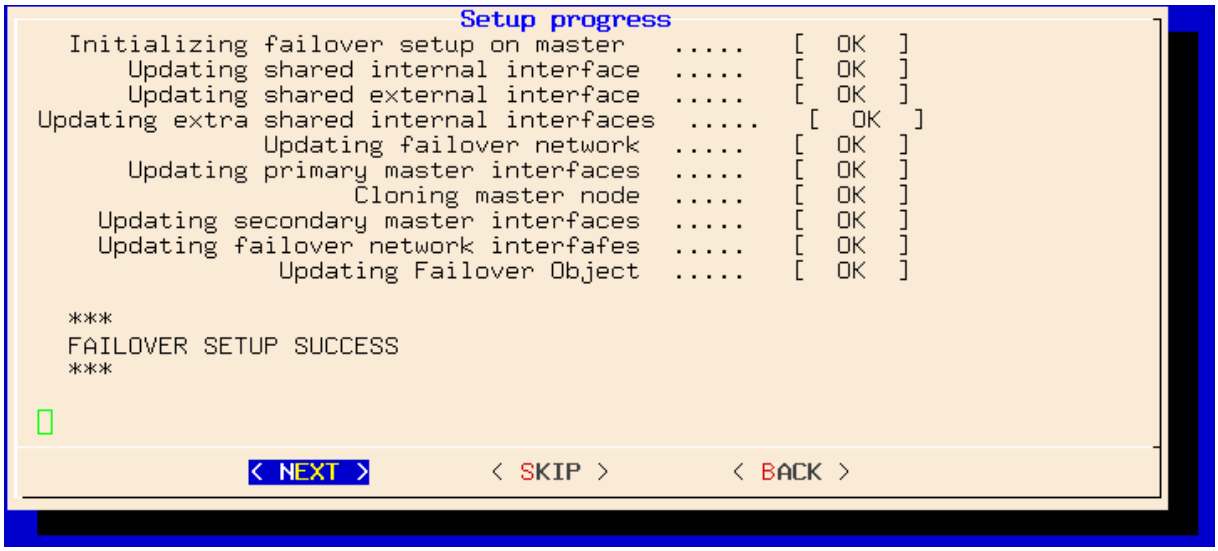


Figure 15.3: `cmha-setup` Setup Progress For Planned Configuration

18. Instructions on what to run on the secondary to clone it from the primary are displayed (figure 15.4).

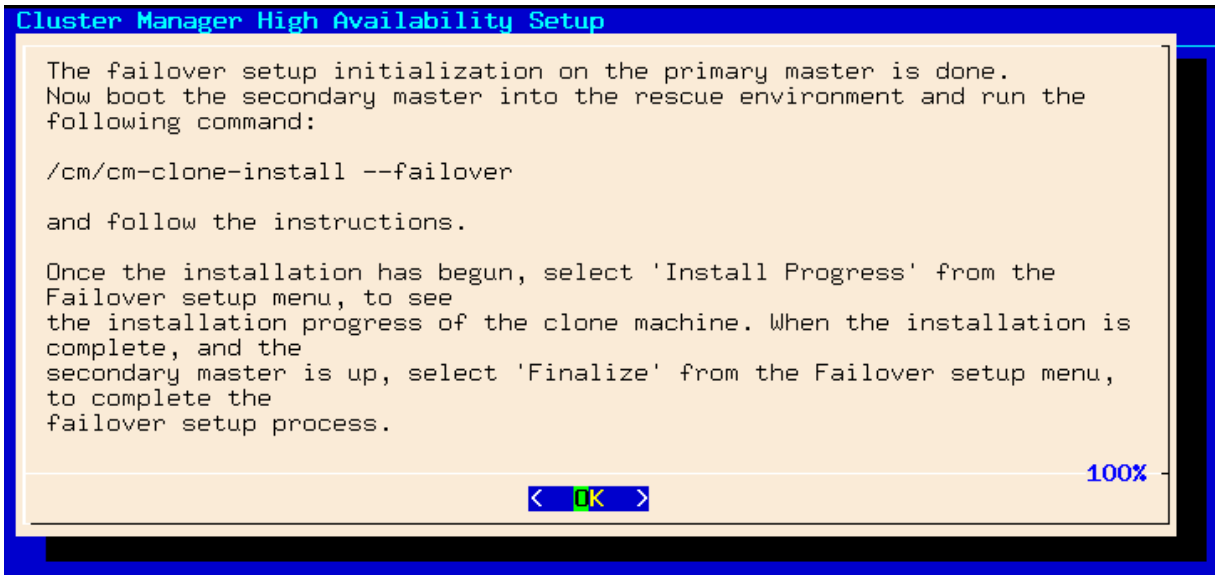


Figure 15.4: `cmha-setup` Instructions To Run On Secondary For Cloning

15.2.2 Failover Cloning (Replacing A Passive Head)

In the IT industry, if an image is made of a computer, then it means making a copy of the drive. In BCM the word "image" is normally used for the software that can be placed on regular nodes. So, BCM uses the word "cloning" to describe making a very similar, or even identical, copy of a head node, using the `/cm/cm-clone-install` command.

There are actually two kinds of cloning possible with the `/cm/cm-clone-install` command:

- **Failover cloning**: With this, a passive head node can be created from the active head node. This uses the `--failover` option to create a copy that is very similar to the active head node, but with changes to make it a passive head, ready for failover purposes, and replacing a head that has just failed.

- **Re-cloning**: An active head node can be created from the active head node. This uses the `--clone` option to create an exact copy (re-clone) of the head node. This might be useful if for some reason the administrator would like to take a snapshot of the head node at that moment. Using this snapshot to have a plug-in replacement head node—that is, a head node kept aside, and ready to replace a failed production head node later on—is not recommended, due to how impractical it is to update the snapshot.

The process described in this section PXE boots the passive from the active, thereby loading a special rescue image from the active that allows cloning from the active to the passive to take place. This section is therefore about failover cloning. How to carry out re-cloning is described in section 15.4.8.

After the preparation has been done by configuring parameters as outlined in section 15.2.1, the failover cloning of the head nodes is carried out. In the cloning instructions that follow, the active node refers to the primary node and the passive node refers to the secondary node. However this correlation is only true for when an HA setup is created for the first time, and it is not necessarily true if head nodes are replaced later on by cloning.

These cloning instructions may also be repeated later on if a passive head node ever needs to be replaced, for example, if the hardware is defective (section 15.4.8). In that case the active head node can be either the primary or secondary.

NOTE: On an **AArch64 (ARM64)** secondary head node, the PXE boot in the following steps, at step 1, can fail before the `Cluster Manager PXE Environment` menu even appears. The generated PXE configuration can advertise a nonexistent `aarch64/syslinux.efi` boot file—a PXELINUX artifact that is not part of the ARM64 TFTP payload—instead of the boot loader that ARM64 head nodes actually ship, `aarch64/grub2.efi`. The passive head node then receives a TFTP "file not found" error and never reaches the rescue environment.

If this occurs, the following `cmsh` commands, run against the secondary head node's device object from the active head node, resolve it:

Example

```
[basecm11]% device
[basecm11->device]% use master2
[basecm11->device[master2]]% set -e grub yes
[basecm11->device*[master2*]]% set -e bootloaderprotocol tftp
[basecm11->device*[master2*]]% commit
```

Both properties must be set, in the same commit. Once committed, the PXE boot in step 1 proceeds as described.

1. The passive head node is PXE booted off the internal cluster network, from the active head node. It is highly recommended that the active and passive head nodes have identical hardware configurations. The BIOS clock of the head nodes should match and be set to the local time. Typically, the BIOS of both head nodes is also configured so that a hard disk boot is attempted first, and a PXE

boot is attempted after a hard disk boot failure, leading to the `Cluster Manager PXE Environment` menu of options. This menu has a 5s time-out.

2. In the `Cluster Manager PXE Environment` menu of the node that is to become a clone, before the 5s time-out, "`Start Rescue Environment`" is selected to boot the node into a Linux ramdisk environment.

3. Once the rescue environment has finished booting, a login as root is done. No password is required (figure 15.5).

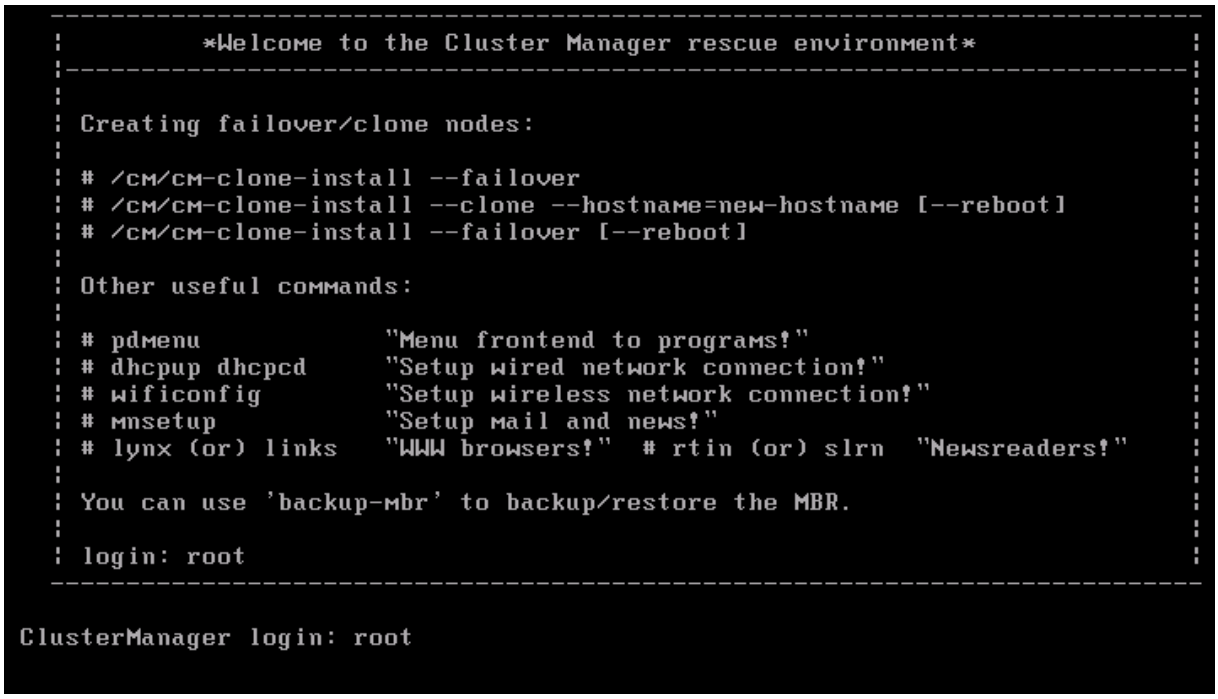


Figure 15.5: Login Screen After Booting Passive Into Rescue Mode From Active

4. The following command is executed (figure 15.6) on the node that is to become a failover clone:
`/cm/cm-clone-install --failover`

   When doing a re-clone as in section 15.4.8, instead of a failover clone, then it is the `--clone` option that is used instead of the `--failover` option.

```
ClusterManager login: root
Welcome to Linux 2.6.32-220.23.1.el6.x86_64.
No mail.
# /cm/cm-clone-install --failover
Network interface to use [default: eth0]:
Please wait while authentication is being set up....
root@master's password:
Please wait while installation begins...
Verifying license              ..............  [  OK  ]
Getting build config           ..............  [  OK  ]
Getting disk layout            ..............  [  OK  ]
The head node disk layout is saved in /cm/__masterdisksetup.xml
[v - view, e - edit, c - continue ]: c
The contents of the following disks will be erased.
/dev/sda
Do you want to continue [yes/no]? yes
Getting mount points           ..............  [  OK  ]
Partitioning hard drive        ..............  [  OK  ]
Syncing hard drive             ..............  [  OK  ]
Finalizing installation        ..............  [  OK  ]
Do you want to reboot[y/n]:_
```

Figure 15.6: Cloning The Passive From The Active Via A Rescue Mode Session

5. When prompted to enter a network interface to use, the interface that was used to boot from the internal cluster network (e.g. `eth0`, `eth1`, ...) is entered. There is often uncertainty about what interface name corresponds to what physical port. This can be resolved by switching to another console and using "`ethtool -p <interface>`", which makes the NIC corresponding to the interface blink.

6. If the provided network interface is correct, a `root@master's password` prompt appears. The administrator should enter the root password.

7. An opportunity to view or edit the master disk layout is offered.

8. A confirmation that the contents of the specified disk are to be erased is asked for.

9. The cloning takes place. The "syncing" stage usually takes the most time. Cloning progress can also be viewed on the active by selecting the "`Install Progress`" option from the `Setup` menu. When viewing progress using this option, the display is automatically updated as changes occur.

10. After the cloning process has finished, a prompt at the console of the passive asks if a reboot is to be carried out. A "`y`" is typed in response to this. The passive node should be set to reboot off its hard drive. This may require an interrupt during reboot, to enter a change in the BIOS setting, if for example, the passive node is set to network boot first.

11. Continuing on now on the active head node, `Finalize` is selected from the `Setup` menu of `cmha-setup`.

12. The MySQL root password is requested. After the password is entered, the administrator is asked whether to replicate all non-system MySQL databases (section 15.1.3). The progress of the `Finalize` procedure is then displayed, and the cloning procedure continues.

13. The cloning procedure of `cmha-setup` pauses to offer the option to reboot the passive. The administrator should accept the reboot option. After reboot, the cloning procedure is complete. The administrator can then go to the `main` menu and `quit` from there or go on to configure "`Shared Storage`" (section 15.2.3) from there.

A check can already be done at this stage on the failover status of the head nodes with the `cmha` command, run from either head node:

Example

```
[root@basecm11 ~]# cmha status
Node Status: running in active master mode

Failover status:
basecm11* -> head2
  failoverping [  OK  ]
  mysql        [  OK  ]
  ping         [  OK  ]
  status       [  OK  ]
head2 -> basecm11*
  failoverping [  OK  ]
  mysql        [  OK  ]
  ping         [  OK  ]
  status       [  OK  ]
```

Here, the asterisk indicates the active node, and the arrow direction indicates which node was carrying out the status check on the other. The `[OK]` states for `mysql`, `ping` and `status` indicate that HA setup completed successfully. The `failoverping` state uses the dedicated failover network route for its checks, and starts working as soon as the passive head node has been rebooted.

15.2.3 Shared Storage Setup

After cloning the head node (section 15.2.2), the last basic stage of creating an HA setup is setting up shared storage.

NAS

1. In the `cmha-setup` main menu, the "`Shared Storage`" option is selected.
2. `NAS` is selected.
3. The parts of the head node filesystem that are to be copied to the NAS filesystems are selected. By default, these are `/home` and `/cm/shared` as suggested in section 15.1.5. The point in the filesystem where the copying is done is the future mount path to where the NAS will share the shared filesystem.

   An already-configured export that is not shared is disabled in `/etc/exports` by `cmha-setup`. This is done to prevent the creation of stale NFS file handles during a failover. Sharing already-existing exports is therefore recommended. Storage can however be dealt with in a customized manner with mount and unmount scripts (page 804).
4. The NFS host name is configured. Also, for each head node filesystem that is to be copied to the NAS filesystem, there is an associated path on the NAS filesystem where the share is to be served from. These NFS volume paths are now configured.
5. If the configured NFS filesystems can be correctly mounted from the NAS server, the process of copying the local filesystems onto the NAS server begins.

15.2.4 Automated Failover And Relevant Testing

A power-off operation on the active head node server does not mean the same as just pulling out the power cable to the active head node. These actions typically have different effects, and should therefore not be confused with each other. During the power-off operation, the power to the main system is switched off, but the BMC remains up. However, in the case of pulling out the power cable, the BMC is typically turned off too. If the BMC is not reachable, then it means that verifying that the active head has been terminated is uncertain. This is because the data that CMDaemon can access implies a logical

possibility that there is a network failure rather than a head node failure. CMDaemon therefore does not carry out an automatic failover if the power cable is pulled out.

For automatic failover to work, the two head nodes must be able to power off their counterpart. This is done by setting up power control (Chapter 4).

Testing If Power Control Is Working

The "device power status" command in cmsh can be used to verify that power control is functional:

Example

```
[master1]% device power status -n mycluster1,mycluster2
 apc03:21 ............ [   ON    ] mycluster1
 apc04:18 ............ [   ON    ] mycluster2
```

Testing The BMC Interface Is Working

If a BMC (Baseboard Management Controller, section 3.7) such as IPMI or iLO is used for power control, it is possible that a head node is not able to reach its own BMC interface over the network. This is especially true when no dedicated BMC network port is used. In this case, cmsh -c "device power status" reports a failure for the active head node. This does not necessarily mean that the head nodes cannot reach the BMC interface of their counterpart. Pinging a BMC interface can be used to verify that the BMC interface of a head node is reachable from its counterpart.

Example

Verifying that the BMC interface of mycluster2 is reachable from mycluster1:

```
[root@mycluster1 ~]# ping -c 1 mycluster2.bmc.cluster
PING mycluster2.bmc.cluster (10.148.255.253) 56(84) bytes of data.
64 bytes from mycluster2.bmc.cluster (10.148.255.253): icmp_seq=1
ttl=64 time=0.033 ms
```

Verifying that the BMC interface of mycluster1 is reachable from mycluster2:

```
[root@mycluster2 ~]# ping -c 1 mycluster1.bmc.cluster
PING mycluster1.bmc.cluster (10.148.255.254) 56(84) bytes of data.
64 bytes from mycluster1.bmc.cluster (10.148.255.254): icmp_seq=1
ttl=64 time=0.028 ms
```

Testing Automated Failover Against A Simulated Crash

A normal (graceful) shutdown of an active head node, does not cause the passive to become active, because HA assumes a graceful failover means there is no intention to trigger a failover. To carry out testing of an HA setup with automated failover, it is therefore useful to simulate a kernel crash on one of the head nodes. The following command crashes a head node instantly:

```
echo c > /proc/sysrq-trigger
```

After the active head node freezes as a result of the crash, the passive head node powers off the machine that has frozen and switches to active mode. A hard crash like this can cause a database replication inconsistency when the crashed head node is brought back up and running again, this time passively, alongside the node that took over. This is normally indicated by a FAILED status for the output of cmha status for MySQL (section 15.4). Database administration with the dbreclone command (section 15.4) may therefore be needed to synchronize the databases on both head nodes to a consistent state. Because dbreclone is a resource-intensive utility, it is best used during a period when there are few or no users. It is generally only used by administrators when they are instructed to do so by BCM support.

A passive node can also be made active without a crash of the active-until-then node, by using the "cmha makeactive" command on the passive (section 15.4.2). Manually running this is not needed in the case of a head node crash in a cluster where power management has been set up for the head nodes, and the automatic failover setting is not disabled.

15.3 Running cmha-setup Without ncurses, Using An XML Specification

15.3.1 Why Run It Without ncurses?

The ncurses-based TUI for cmha-setup is normally how administrators should set up a failover configuration.

The express mode of cmha-setup is the command-line interface (CLI) that allows an administrator to skip the TUI. This is useful, for example, for scripting purposes and speeding deployment. A further convenience is that this mode uses a human-editable XML file to specify the network and storage definitions for failover.

Running cmha-setup without the TUI still requires some user intervention, such as entering the root password for MySQL. The intervention required is scriptable with, for example, Expect, and is minimized if relevant options are specified for cmha-setup from the -x options.

15.3.2 The Syntax Of cmha-setup Without ncurses

The express mode (-x) options are displayed when "cmha-setup -h" is run. The syntax of the -x options is indicated by:

```
cmha-setup [ -x -c <configfile> [-s <type>] <-i|-f[-r]> [-p <mysqlrootpassword>] [-a] ]
```

The -x options are:

- -c|--config <*configfile*>: specifies the location of <*configfile*>, which is the failover configuration XML file for cmha-setup. The file stores the values to be used for setting up a failover head node. The recommended location is at /cm/local/apps/cluster-tools/ha/conf/failoverconf.xml.
- -i|--initialize: prepares a failover setup by setting values in the CMDaemon database to the values specified in the configuration file. This corresponds to section 15.2.1. The administrator is prompted for the MySQL root password unless the -p option is used. The -i option of the script then updates the interfaces in the database, and clones the head node in the CMDaemon database. After this option in the script is done, the administrator normally clones the passive node from the active, as described in steps 1 to 10 of section 15.2.2.
- -f|--finalize: After the passive node is cloned as described in steps 1 to 10 of section 15.2.2, the finalize option is run on the active node to run the non-TUI finalize procedure. This is the non-TUI version of steps 11 to 13 of section 15.2.2.
  - -r|--finalizereboot: makes the passive reboot after the finalize step completes.
- -p|--pass <*mysqlrootpassword*>: specifies the MySQL root password. Leaving this out means the administrator is prompted to type in the password during a run of the cmha-setup script when using the -x options.
- -a|--replicate-all-databases: replicates all non-system MySQL databases and users/grants to the secondary head node, instead of only cmdaemon and slurm_acct_db (section 15.1.3).

There is little attempt at validation with the express mode, and invalid entries can cause the command to hang.

15.3.3 Example cmha-setup Run Without ncurses

Preparation And Initialization:

After shutting down all nodes except for the active head node, a configuration is prepared by the administrator in /cm/local/apps/cluster-tools/ha/conf/failoverconf.xml. The administrator then runs cmha-setup with the initialization option on the active:

```
[root@basecm11 ~]# cd /cm/local/apps/cluster-tools/ha/conf
[root@basecm11 conf]# cmha-setup -x -c failoverconf.xml -i
Please enter the mysql root password:
   Initializing failover setup on master   .....    [  OK  ]
       Updating shared internal interface  .....    [  OK  ]
       Updating shared external interface  .....    [  OK  ]
Updating extra shared internal interfaces  .....    [  OK  ]
                Updating failover network  .....    [  OK  ]
       Updating primary master interfaces  .....    [  OK  ]
                      Cloning master node  .....    [  OK  ]
     Updating secondary master interfaces  .....    [  OK  ]
     Updating failover network interfaces  .....    [  OK  ]
                 Updating Failover Object  .....    [  OK  ]
```

The preceding corresponds to the steps in section 15.2.1.

PXE Booting And Cloning The Passive:
The passive head node is then booted up via PXE and cloned as described in steps 1 to 10 of section 15.2.2.

Finalizing On The Active And Rebooting The Passive:
Then, back on the active head node the administrator continues the session there, by running the finalization option with a reboot option:

```
[root@basecm11 conf]# cmha-setup -x -c failoverconf.xml -f -r
Please enter the mysql root password:
    Updating secondary master mac address  .....    [  OK  ]
   Initializing failover setup on head2  .....    [  OK  ]
                         Cloning database  .....    [  OK  ]
                    Update DB permissions  .....    [  OK  ]
  Checking for dedicated failover network  .....    [  OK  ]
A reboot has been issued on head2
```

The preceding corresponds to steps 11 to 13 of section 15.2.2.

Adding Storage:
Continuing with the session on the active, setting up a shared storage could be done with:

```
[root@basecm11 conf]# cmha-setup -x -c failoverconf.xml -s nas
```

The preceding corresponds to carrying out the NAS procedure of section 15.2.3.

15.4 Managing HA

Once an HA setup has been created, the tools in this section can be used to manage the HA aspects of the cluster.

15.4.1 Changing An Existing Failover Configuration

Changing an existing failover configuration is usually done most simply by running through the HA setup procedure of section 15.2 again, with one exception. The exception is that the existing failover configuration must be removed by using the "`Undo Failover`" menu option between steps 3 and 4 of the procedure described in section 15.2.1.

15.4.2 `cmha` Utility

A major command-line utility for interacting with the HA subsystem, for regular nodes as well as for head nodes, is `cmha`. It is part of the BCM `cluster-tools` package. Its usage information is:

```
[root@mycluster1 ~]# cmha
Usage: cmha < status | makeactive [node] | dbreclone <host> |
  nodestatus [name] >


status              Retrieve and print high availability status
                    of head nodes.

nodestatus [groups] Retrieve and print high availability status
                    of failover [groups] (comma separated list of group
                    names. If no argument is given, then the status of
                    all available failover groups is printed.

makeactive [node]   Make the current head node the active head node. If
                    [node] is specified, then make [node] the active
                    node in the failover group that [node] is part of.

dbreclone <host>    Clone MySQL database from this head node to
                    <host> (hostname of failover head node).
```

Some of the information and functions of `cmha` can also be carried out via CMDaemon:

- For `cmsh`, the following commands can be run from within the `base` object in `partition` mode:
  - For the head node, the `status` and `makeactive` commands are run from within the `failover` submode.
  - For regular nodes the `nodestatus` and `makeactive` [*node*] commands are run from within the `failovergroups` submode.

The `dbreclone` option cannot be carried out in Base View or `cmsh` because it requires stopping CMDaemon.

The `cmha` options `status`, `makeactive`, and `dbreclone` are looked at in greater detail next:

`cmha status`: Querying HA Status

Information on the failover status is displayed thus:

Example

```
[root@mycluster1 ~]# cmha status
Node Status: running in active master mode

Failover status:
mycluster1* -> mycluster2
  failoverping [  OK  ]
  mysql        [  OK  ]
  ping         [  OK  ]
  status       [  OK  ]
mycluster2 -> mycluster1*
  failoverping [  OK  ]
  mysql        [  OK  ]
  ping         [  OK  ]
  status       [  OK  ]
```

The `*` in the output indicates the head node which is currently active. The `status` output shows 4 aspects of the HA subsystem from the perspective of each head node:

HA Status	Description
failoverping	the other head node is reachable via the dedicated failover network. This failover ping uses the failover route instead of the internal net route. It uses ICMP ping
mysql	MySQL replication status
ping	the other head node is reachable over the primary management network. It uses ICMP ping.
status	CMDaemon running on the other head node responds to REST calls

By default, BCM prepares to carry out the failover sequence (the sequence that includes a STONITH) when all three of `ping`, `failoverping` and `status` are not `OK` on a head node. If these three are not OK, then the active node is *all dead* according to `cmha`. One way of initiating failover is thus by causing a system crash (section 15.2.4).

It can typically take about 30s for the `cmha status` command to output its findings in the case of a recently crashed head node.

`cmha makeactive`: Initiate Failover

If automatic failover is enabled (section 15.1.7), then the failover sequence attempts to complete automatically if power management is working properly, and the `cmha status` shows `ping`, `failoverping` and `status` as failed.

If automatic failover is disabled, then a manual failover operation must be executed to have a failover operation take place. A manual failover operation can be carried out with the "`cmha makeactive`" command:

Example

To initiate a failover manually:

```
[root@mycluster2 ~]# cmha makeactive
Proceeding will initiate a failover sequence which will make this node
(mycluster2) the active master.

Are you sure ? [Y/N]
y
Your session ended because: CMDaemon failover, no longer master
mycluster2 became active master, reconnecting your cmsh ...
```

On successful execution of the command, the former active head node simply continues to run as a passive head node.

The `cmha makeactive` command assumes both head nodes have no problems preventing the execution of the command.

One possible problem that can halt manual failover is if nodes are being provisioned by the provisioning subsystem at that time (section 5.2.4). In that case, provisioning should be cancelled by the cluster administrator before the `cmha makeactive` command can continue, for example, with `cmsh -c "softwareimage cancelprovisioningrequest -a"` (page 255).

For automatic failover no such intervention takes place—provisioning requests are killed when the active head node is powered off.

Another slightly similar problem that can occur for automatic failover, as well as manual failover, is the "mostly dead" edge case. This case requires careful consideration before the `Are you sure ?` prompt is answered by the cluster administrator.

`cmha makeactive` edge case—the mostly dead active:

- For a manual failover operation, if the execution of the `cmha makeactive` command has problems, then it can mean that there is a problem with the initially active head node being in a sluggish state. That is, neither fully functioning, nor all dead. The active head node is thus in a state that is still powered on, but what can be called *mostly dead*. There is a big difference between mostly dead and *all dead*. Mostly dead means slightly alive (not all of `ping`, `failoverping`, and `status` are FAILED), while all dead means there is only one thing that can sensibly be done to make sure the cluster keeps running—that is, to make the old passive the new active.

  Making an old passive the new active is only safe if the old active is guaranteed to not come back as an active head node. This guarantee is set by a STONITH (page 805) for the old active head node, and results in a former active that is now all dead. STONITH thus guarantees that head nodes are not in conflict about their active and passive states. STONITH can however still fail in achieving a clean shutdown when acting on a mostly dead active head node, which can result in unclean filesystem or database states.

  Thus, the mostly dead active head node may still be in the middle of a transaction, so that shutting it down may cause filesystem or database corruption. Making the passive node also active then in this case carries risks such as mounting filesystems accidentally on both head nodes, or carrying out database transactions on both nodes. This can also result in filesystem and database corruption.

  It is therefore left to the administrator to examine the situation for corruption risk. The decision is either to power off a mostly dead head node, i.e. STONITH to make sure it is all dead, or whether to wait for a recovery to take place. When carrying out a STONITH on the mostly dead active head node, the administrator must power it off *before* the passive becomes active for a manual failover to take place with minimal errors. The `cmha dbreclone` option may still be needed to restore a corrupted database after such a power off, after bringing the system back up.

- For an automated failover configuration, powering off the mostly dead active head node is not carried out automatically due to the risk of filesystem and database corruption. A mostly dead active node with automatic failover configuration therefore stays mostly dead either until it recovers, or until the administrator decides to do a STONITH manually to ensure it is all dead. Here, too, the `cmha dbreclone` option may still be needed to restore a corrupted database after such a power off, after bringing the system back up.

`cmha dbreclone`: Cloning The CMDaemon Database

The `dbreclone` option of `cmha` clones the CMDaemon state database from the head node on which `cmha` runs to the head node specified after the option. It is normally run in order to clone the database from the active head node to the passive—running it from the passive to the active can cause a loss of database entries. Running the `dbreclone` option can be used to retrieve the MySQL CMDaemon state database tables, if they are, for example, unsalvageably corrupted on the destination node, and the source node has a known good database state. Because it is resource intensive, it is best run when there are few or no users. It is typically only used by administrators after being instructed to do so by BCM support.

Example

```
[root@basecm11 ~]# cmha status
Node Status: running in active master mode

Failover status:
basecm11* -> head2
  failoverping [  OK  ]
  mysql        [  OK  ]
  ping         [  OK  ]
  status       [  OK  ]
```

```
head2 -> basecm11*
  failoverping [  OK  ]
  mysql        [FAILED] (11)
  ping         [  OK  ]
  status       [  OK  ]
[root@basecm11 ~]# cmha dbreclone head2
Proceeding will cause the contents of the cmdaemon state database on he\
ad2 to be resynchronized from this node (i.e. basecm11 -> head2)

Are you sure ? [Y/N]
Y
Waiting for CMDaemon (3113) to terminate...
[  OK  ]
Waiting for CMDaemon (7967) to terminate...
                                                    [  OK  ]
cmdaemon.dump.8853.sql                      100%  253KB 252.9KB/s   00:00
slurmacctdb.dump.8853.sql                   100%   11KB  10.7KB/s   00:00
Waiting for CMDaemon to start...                    [  OK  ]
Waiting for CMDaemon to start...[  OK  ]
[root@basecm11 ~]# cmha status
Node Status: running in active master mode

Failover status:
basecm11* -> head2
  failoverping [  OK  ]
  mysql        [  OK  ]
  ping         [  OK  ]
  status       [  OK  ]
head2 -> basecm11*
  failoverping [  OK  ]
  mysql        [  OK  ]
  ping         [  OK  ]
  status       [  OK  ]
```

15.4.3 States

The state a head node is in can be determined in three different ways:

1 By looking at the message being displayed at login time.

Example

```
------------------------------------------------------------------------

Node Status: running in active master mode

------------------------------------------------------------------------
```

2 By executing `cmha status`.

Example

```
[root@mycluster ~]# cmha status
Node Status: running in active master mode
...
```

3 By examining `/var/spool/cmd/state`.

There are a number of possible states that a head node can be in:

State	Description
`INIT`	Head node is initializing
`FENCING`	Head node is trying to determine whether it should try to become active
`ACTIVE`	Head node is in active mode
`PASSIVE`	Head node is in passive mode
`BECOMEACTIVE`	Head node is in the process of becoming active
`BECOMEPASSIVE`	Head node is in the process of becoming passive
`UNABLETOBECOMEACTIVE`	Head node tried to become active but failed
`ERROR`	Head node is in error state due to unknown problem

Especially when developing custom mount and unmount scripts, it is quite possible for a head node to go into the `UNABLETOBECOMEACTIVE` state. This generally means that the mount and/or unmount script are not working properly or are returning incorrect exit codes. To debug these situations, it is helpful to examine the output in `/var/log/cmdaemon`. The "`cmha makeactive`" shell command can be used to instruct a head node to become active again.

15.4.4 Failover Action Decisions

A table summarizing the scenarios that decide when a passive head should take over is helpful:

Event on active	Reaction on passive	Reason
Reboot	Nothing	Event is usually an administrator action. To make the passive turn active, an administrator would run "`cmha makeactive`" on it.
Shutdown	Nothing	As above.
Unusably sluggish or freezing system by state pingable with ICMP packets	Nothing	1. Active may still unfreeze. 2. Shared filesystems may still be in use by the active. Concurrent use by the passive taking over therefore risks corruption. 3. Mostly dead head can be powered off by administrator after examining situation (section 15.4.2).

...continues

...continued

Event on active	Reaction on passive	Reason
Become passive in response to "`cmha makeactive`" run on passive	Become active when former active becomes passive	As ordered by administrator
Active dies	Quorum called, may lead to passive becoming new active	Confirms if active head is dead according to other nodes too. If so, then a "power off" command is sent to it. If the command is successful, the passive head becomes the new active head.

15.4.5 Keeping Head Nodes In Sync

Order Of Head Node Updates

When the `cmdaemon` package is updated in a high availability configuration, the update is applied to the active head node first. Once the active head node has been confirmed healthy with `cmha status`, the update is applied to the passive head node. Applying the update to the passive head node first causes the two head nodes to go out of sync.

Example

```
[root@basecm11 ~]# cmha status
Node Status: running in active mode
...
[root@basecm11 ~]# yum update cmdaemon     ##(or apt equivalent)
...
[root@basecm11 ~]# cmha status
Node Status: running in active mode
...
[root@basecm11 ~]# systemctl restart cmd
```

The same order applies when cmdaemon is restarted without a package update.

Note for Ubuntu: If the `cmdaemon` and `mysql` packages are updated together in the same `apt` operation, then the update can stall. This is because of a known bug based on CMDaemon waiting for the MySQL database to become available, while MySQL itself is not yet running because `apt` is still mid-upgrade. Until this known issue is resolved (internal reference BCM-31089), it is recommended that the `mysql` package be updated separately first, and only afterwards should `cmdaemon` and the remaining packages be updated.

What Should Be Kept In Sync?

- It is a best practice to carry out a manual `updateprovisioners` command on the active head node, immediately after a regular node software image change has been made.

  A successful run of the `updateprovisioners` command means that in the event of a failover, the formerly passive head node already has up-to-date regular node software images, which makes further administration simpler.

  The background behind why it is done can be skipped, but it is as follows:

  An image on the passive head node, which is a node with a provisioning role, is treated as an image on any other provisioning node. This means that it eventually synchronizes to a changed

image on the active head node. By default the synchronization happens at midnight, which means images may remain out-of-date for up to 24 hours. The images being in an out-of-date state should be viewed as normal, because the timeout period associated with being in an updated state is only 5 minutes by default.

Since the passive head is a provisioning node, it also means that an attempt to provision regular nodes from it with the changed image will not succeed if it happens too soon after the image change event on the active head. "Too soon" means within the autoupdate period defined by the parameter `dirtyautoupdatetimeout` (page 252).

If on the other hand the autoupdate timeout is exceeded, then by itself this does not lead to the image on the passive head node becoming synchronized with an image from the active head node. Such synchronization only takes place as part of regular housekeeping (at midnight by default). Or it takes place if a regular node sends a provisioning request to the passive head node, which can take place during the reboot of the regular node.

This means that the `provisioningstatus` command commonly shows that the passive head node image is "out of date". This may sound alarming to a cluster administrator. However, before the image gets to be used, it is synced, so in practice the "out of date" warning is not something to be concerned about.

The synchronization logic just described is followed to reduce the load on the head node. The only pitfall in this is the case when an administrator changes an image on the active head node, and then soon after that the passive head node becomes active as part of a failover, without the images having had enough time to synchronize. In that case the formerly passive node ends up with out-of-date software images. That is why it is a best practice to carry out a manual `updateprovisioners` command on the active head immediately after a regular node software image change has been made.

- Changes controlled by CMDaemon are synchronized automatically between the CMDaemon databases to the required extent during failover to the active node.

  If the output of `cmha status` is not `OK`, then it typically means that the CMDaemon databases of the active head node and the passive head node are not synchronized. This situation may be resolved by waiting, typically for several minutes. If the status does not resolve on its own, then this indicates a more serious problem which should be investigated further.

- By default, software images are not stored on shared storage, and are synchronized between the head nodes by CMDaemon.

  However, if images are kept on shared storage, then, within the provisioning role (section 5.2.1), the image-related parameters such as `allimages`, `localimages`, and `sharedimages`, must be adjusted according to the configuration used.

- If filesystem changes are made on an active head node without using CMDaemon (`cmsh` or Base View), and if the changes are outside the shared filesystem, then these changes should normally also be made by the administrator on the passive head node. For example:

  - RPM installations/updates (section 9.2)
  - Applications installed locally
  - Files (such as drivers or values) placed in the `/cm/node-installer/` directory and referred to by `initialize` (section 5.4.5) and `finalize` scripts (section 5.4.11)
  - Any other configuration file changes outside of the shared filesystems

The reason behind not syncing everything automatically is to guarantee that a change that breaks a head node is not accidentally propagated to the passive. This way there is always a running head node.

Otherwise, if automated syncing is used, there is a risk of ending up with two broken head nodes at the same time.

If the cluster is being built on bare metal, then a sensible way to minimize the amount of work to be done is to install a single head cluster first. All packages and applications should then be placed, updated and configured on that single head node until it is in a satisfactory state. Only then should HA be set up as described in section 15.2, where the cloning of data from the initial head node to the secondary is described. The result is then that the secondary node gets a well-prepared system with the effort to prepare it having only been carried out once.

Avoiding Encounters With The Old Filesystems

It should be noted that when the shared storage setup is made, the contents of the shared directories (at that time) are copied over from the local filesystem to the newly created shared filesystems. The shared filesystems are then mounted on the mountpoints on the active head node, effectively hiding the local contents.

Since the shared filesystems are only mounted on the active machine, the old filesystem contents remain visible when a head node is operating in passive mode. Logging into the passive head node may thus confuse users and is therefore best avoided.

Updating Services On The Head Nodes And Associated Syncing

The services running on the head nodes described in section 15.1.3 should also have their packages updated on both head nodes.

For services that have no package-update ordering constraint between the head nodes, such as DHCP, LDAP, NTP and DNS, their packages can be updated on both head nodes using the normal package manager commands, at about the same time.

CMDaemon and MySQL are treated differently. Package updates for these two services must follow the order described in section 15.4.5: the active head node is updated first and confirmed healthy with `cmha status`, and only then is the passive head node updated. This ordering exists because a CMDaemon package update can include a MySQL database schema migration, which needs to be applied on the active head node before it is replicated to the passive head node. Manually stopping the CMDaemon or MySQL service before an update is not recommended. The package manager itself handles restarting the service as part of the update.

The provisioning node service is part of the CMDaemon package. The service updates images from the active head node to all provisioning nodes, including the passive head node, if the administrator runs the command to update provisioners. How to update provisioners is described in section 15.1.3.

For services that migrate across head nodes during failover, such as NFS, or the `sgemaster` it is recommended (but not mandated) to carry out this procedure: the package on the passive node (called the secondary for the sake of this example) is updated to check for any broken package behavior. The secondary is then made active with `cmha makeactive` (section 15.4.2), which automatically migrates users cleanly off from being serviced by the active to the secondary. The package is then updated on the primary. If desired, the primary can then be made active again. The reason for recommending this procedure for services that migrate is that, in case the update has issues, the situation can be inspected somewhat better with this procedure.

15.4.6 High Availability Parameters

There are several HA-related parameters that can be tuned. Accessing these via Base View is described in section 15.4.6. In `cmsh` the settings can be accessed in the `failover` submode of the `base` partition.

Example

```
[mycluster1]% partition failover base
[mycluster1->partition[base]->failover]% show
Parameter                      Value
------------------------------ ----------------------------
```

```
Dead time                       10
Disable automatic failover      no
Failover network                failovernet
Init dead                       30
Keep alive                      1
Mount script
Postfailover script
Prefailover script
Quorum time                     60
Revision
Secondary headnode
Unmount script
Warn time                       5
```

`Dead time`
When a passive head node determines that the active head node is not responding to any of the periodic checks for a period longer than the `Dead time` seconds, the active head node is considered dead and a quorum procedure starts. Depending on the outcome of the quorum, a failover sequence may be initiated.

`Disable automatic failover`
Setting this to yes disables automated failover. Section 15.1.7 covers this further.

`Failover network`
The `Failover network` setting determines which network is used as a dedicated network for the `failoverping` heartbeat check. The heartbeat connection is normally a direct cable from a NIC on one head node to a NIC on the other head node. The network can be selected via tab-completion suggestions. By default, without a dedicated failover network, the possibilities are nothing, `externalnet` and `internalnet`.

`Init dead`
When head nodes are booted simultaneously, the standard `Dead time` might be too strict if one head node requires a bit more time for booting than the other. For this reason, when a head node boots (or more exactly, when the cluster management daemon is starting), a time of `Init dead` seconds is used rather than the `Dead time` to determine whether the other node is alive.

`Keep alive`
The `Keep alive` value is the time interval, in seconds, over which the passive head node carries out a check that the active head node is still up. If a dedicated failover network is used, 3 separate heartbeat checks are carried out to determine if a head node is reachable.

`Mount script`
The script pointed to by the `Mount script` setting is responsible for bringing up and mounting the shared filesystems.

`Postfailover script`
The script pointed to by the `Postfailover script` setting is run by `cmdaemon` on both head nodes. The script first runs on the head that is now passive, then on the head that is now active. It runs as soon as the former passive has become active. It is typically used by scripts mounting an NFS shared storage so that no more than one head node exports a filesystem to NFS clients at a time.

`Prefailover script`
The script pointed to by the `Prefailover script` setting is run by `cmdaemon` on both head nodes. The script first runs on the (still) active head, then on the (still) passive head. It runs as soon as the decision

for the passive to become active has been made, but before the changes are implemented. It is typically used by scripts unmounting an NFS shared storage so that no more than one head node exports a filesystem to NFS clients at a time. When unmounting shared storage, it is very important to ensure that a non-zero exit code is returned if unmounting has problems, or the storage may become mounted twice during the `Postfailover script` stage, resulting in data corruption.

`Quorum time`
When a node is asked what head nodes it is able to reach over the network, the node has `Quorum time` seconds to respond. If a node does not respond to a call for quorum within that time, it is no longer considered for the results of the quorum check.

`Secondary headnode`
The `Secondary headnode` setting is used to define the secondary head node to the cluster.

`Unmount script`
The script pointed to by the `Unmount script` setting is responsible for bringing down and unmounting the shared filesystems.

`Warn time`
When a passive head node determines that the active head node is not responding to any of the periodic checks for a period longer than `Warn time` seconds, a warning is logged that the active head node might become unreachable soon.

15.4.7 Viewing Failover Using Base View

Accessing `cmsh` HA Parameters (`partition failover base`) Using Base View
The Base View equivalents of the `cmsh` HA parameters in section 15.4.6 are accessed from the navigation path:

```
Cluster > Partition[base] > Settings > Failover
```

15.4.8 Re-cloning A Head Node

Some time after an HA setup has gone into production, it may become necessary to re-install one of the head nodes, for example if one of the head nodes were replaced due to hardware failure.

To re-clone a head node from an existing active head node, the head node hardware that is going to become the clone can be PXE-booted into the rescue environment, as described in section 15.2.2. Instead of running the `cm-clone-install --failover` command as in that section, the following command can be run:

```
[root@basecm11 ~]# /cm/cm-clone-install --clone --hostname=<new host name>
```

The new host name can be the same as the original, because the clone is not run at the same time as the original anyway. The clone should not be run after cloning on the same network segment as the original, in order to prevent IP address conflicts.

If the clone is merely intended as a backup, then the clone hardware does not have to match the head node. For a backup, typically the most important requirement is then that a clone drive should not run out of space—that is, its drive should be as large as, or larger than the matching drive of the head node.

If the clone is to be put to work as a head node, then, if the MAC address of one of the head nodes has changed, it is typically necessary to request that the product key is unlocked, so that a new license can be obtained (section 4.3 of the *Installation Manual*).

Also, for a clone that is to be put to work as a head node, the CMDaemon database should first be synchronized from the active head node to the clone. This can be done by running `cmha dbreclone` on the active head node (page 820) before carrying out tasks with `cmsh` or Base View.

Exclude Lists And Cloning

Some files are normally excluded from being copied across from the head node to the clone, because syncing them is not appropriate.

The following exclude files are read from inside the directory `/cm/` on the clone node when the `cm-clone-install` command is run (step 4 in section 15.2.2).

- `excludelistnormal`: used to exclude files to help generate a clone of the other head node. It is read when running the `cm-clone-install` command without the `--failover` option.
- `excludelistfailover`: used to exclude files to help generate a passive head node from an active head node. It is read when running the `cm-clone-install --failover` command.

In a default cluster, there is no need to alter these exclude files. However some custom head node configurations may require appending a path to the list.

The cloning that is carried out is logged in `/var/log/clone-install-log`.

Exclude Lists In Perspective

The exclude lists `excludelistfailover` and `excludelistnormal` described in the preceding paragraphs should not be confused with the exclude lists of section 5.6.1. The exclude lists of section 5.6.1:

- `excludelistupdate`
- `excludelistfullinstall`
- `excludelistsyncinstall`
- `excludelistgrabnew`
- `excludelistgrab`
- `excludelistmanipulatescript`

are Base View or `cmsh` options, and are maintained by CMDaemon. On the other hand, the exclude lists introduced in this section (15.4.8):

- `excludelistfailover`
- `excludelistnormal`

are not Base View or `cmsh` options, are not modified with `excludelistmanipulatescript`, are not maintained by CMDaemon, but are made use of when running the `cm-clone-install` command.

Btrfs And `cm-clone-install`

If a partition with Btrfs (section 14.4.1) is being cloned using `cm-clone-install`, then by default only mounted snapshots are cloned.

If all the snapshots are to be cloned, then the `--btrfs-full-clone` flag should be passed to the `cm-clone-install` command. This flag clones all the snapshots, but it is carried out with duplication (bypassing the COW method), which means the filesystem size can increase greatly.

15.5 HA For Regular Nodes And Edge Director Nodes

HA for regular nodes is available from NVIDIA Base Command Manager version 7.0 onward. HA for edge director nodes follows a similar design to HA for regular nodes, and is available from NVIDIA Base Command Manager version 9.2 onward.

15.5.1 Why Have HA On Non-Head Nodes?

Why Have HA On Regular Nodes?

HA for regular nodes can be used to add services to the cluster and make them HA. Migrating the default existing HA services that run on the head node is not recommended, since these are optimized and integrated to work well as is. Instead, good candidates for this feature are other, extra, services that can run, or are already running, on regular nodes, but which benefit from the extra advantage of HA.

Why Have HA On Edge Director Nodes?

Edge director nodes manage edge nodes in a similar way to how head nodes manager regular nodes. Edge directors therefore benefit from HA for the same main that head nodes benefit from HA: avoiding a single point of failure (section 15.0.1).

15.5.2 Comparing HA For Head Nodes, Regular Nodes And Edge Director Nodes

Many of the features of HA for regular nodes and edge director nodes are as HA for head nodes. These include:

HA For Head Nodes, Regular Nodes, And Edge Director Nodes: Some Features In Common

Power control is needed for all HA nodes, in order to carry out automatic failover (section 15.1.7).

Warn time and dead time parameters can be set (section 15.4.6).

Mount and unmount scripts (page 804).

Pre- and post- failover scripts (section 15.4.6).

Disabling or enabling automatic failover (section 15.1.7).

A virtual shared IP address that is presented as the virtual node that is always up (section 15.1.4).

Some differences between head node HA and the other types of HA are:

HA For Head Nodes, Regular Nodes, And Edge Director Nodes: Some Features That Differ		
Head Node HA	**Regular Node HA**	**Edge Director Node HA**
Installed with `cmha-setup` (section 15.2).	Installed by administrator using a procedure similar to section 15.5.3.	Installed by administrator using `cm-edge-setup` (section 2.1.1 of the *Edge Manual*)
Configurable within `failover` submode of `partition` mode (section 15.4.6).	Configurable within `failovergroups` submode of `partition` mode (section 15.5.3).	Configurable within `failovergroups` submode of `partition` mode (section 15.5.3). However, manual configuration should not be carried out. Instead, the settings should be configured during installation with the `cm-edge-setup` utility.
Only one passive node.	Multiple passive nodes defined by failover groups.	Multiple passive nodes defined by failover groups.
Can use the optional failover network and `failoverping` heartbeat.	No failover network. Heartbeat checks done via regular node network.	No failover network. Heartbeat checks done via regular node network.
A quorum procedure (section 15.1.6). If more than half the nodes can only connect to the passive, then the passive powers off the active and becomes the new active.	Active head node does checks. If active regular node is apparently dead, it is powered off (STONITH). Another regular node is then made active.	Active head node does checks. If active edge director node is apparently dead, it is powered off (STONITH). Another edge director node is then made active.

Failover Groups

Regular nodes use *failover groups* to identify nodes that are grouped for HA. Two or more nodes are needed for a failover group to function. During normal operation, one member of the failover group is active, while the rest are passive. A group typically provides a particular service.

Edge directors also use failover groups for HA. However configuration of edge directors is generally best done during installation with `cm-edge-setup` (section 2.1.1 of the *Edge Manual*).

15.5.3 Setting Up A Regular Node HA Service

In `cmsh` a regular node HA service, CUPS in this example, can be set up as follows:

Making The Failover Group

A failover group must first be made, if it does not already exist:

Example

```
[basecm11->partition[base]->failovergroups]% status
No active failover groups
[basecm11->partition[base]->failovergroups]% add cupsgroup
[basecm11->partition*[base*]->failovergroups*[cupsgroup*]]% list
Name (key)               Nodes
------------------------ ------------------------
cupsgroup
```

By default, CUPS is provided in the standard image, in a stopped state. In Base View a failover group can be added via the navigation path `Cluster` > `Partition[base]` > `Settings` > `Failover groups` > `Add`

Adding Nodes To The Failover Group

Regular nodes can then be added to a failover group. On adding, BCM ensures that one of the nodes in the failover group becomes designated as the active one in the group (some text elided):

Example

```
[basecm11->...[cupsgroup*]]% set nodes node001..node002
[basecm11->...[cupsgroup*]]% commit
[basecm11->...[cupsgroup]]%
...Failover group cupsgroup, make node001 become active
...Failover group cupsgroup, failover complete. node001 became active
[basecm11->partition[base]->failovergroups[cupsgroup]]%
```

Setting Up A Server For The Failover Group

The CUPS server needs to be configured to run as a service on all the failover group nodes. The usual way to configure the service is to set it to run only if the node is active, and to be in a stopped state if the node is passive:

Example

```
[basecm11->partition[base]->failovergroups[cupsgroup]]% device
[basecm11->device]% foreach -n node001..node002 (services; add cups; \
set runif active; set autostart yes; set monitored yes)
[basecm11->device]% commit
Successfully committed 2 Devices
[basecm11->device]%
Mon Apr  7 08:45:54 2014 [notice] node001: Service cups was started
```

The `runif` options are described in section 3.15.1.

Setting And Viewing Parameters And Status In The Failover Group

Knowing which node is active: The status command shows a summary of the various failover groups in the `failovergroups` submode, including which node in each group is currently the active one:

Example

```
[basecm11->partition[base]->failovergroups]% status
Name         State        Active        Nodes
------------ ------------ ------------ --------------------------
cupsgroup    ok           node001      node001,node002 [   UP   ]
```

Making a node active: To set a particular node to be active, the `makeactive` command can be used from within the failover group:

Example

```
[basecm11->partition[base]->failovergroups]% use cupsgroup
[basecm11->...]->failovergroups[cupsgroup]]% makeactive node002
node002 becoming active ...
[basecm11->partition[base]->failovergroups[cupsgroup]]%
... Failover group cupsgroup, make node002 become active
...node001: Service cups was stopped
...node002: Service cups was started
...Failover group cupsgroup, failover complete. node002 became active
```

An alternative is to simply use the `cmha` utility (section 15.4.2):

Example

```
[root@basecm11 ~]# cmha makeactive node002
```

Parameters for failover groups: Some useful regular node HA parameters for the failover group object, cupsgroup in this case, can be seen with the show command:

Example

```
[basecm11->partition[base]->failovergroups]% show cupsgroup
Parameter                                  Value
------------------------------------------ --------------------
Automatic failover after graceful shutdown no
Dead time                                  10
Disable automatic failover                 no
Mount script
Name                                       cupsgroup
Nodes                                      node001,node002
Postfailover script
Prefailover script
Revision
Unmount script
Warn time                                  5
```

Setting Up The Virtual Interface To Make The Server An HA Service

The administrator then assigns each node in the failover group the same alias interface name and IP address dotted quad on its physical interface. The alias interface for each node should be assigned to start up if the node becomes active.

Example

```
[basecm11->device]% foreach -n node001..node002 (interfaces; add alias \
bootif:0 ; set ip 10.141.240.1; set startif active; set network internalnet)
[basecm11->device*]% commit
Successfully committed 2 Devices
[basecm11->device]% foreach -n node001..node002 (interfaces; list)
Type         Network device name  IP              Network
------------ -------------------- --------------- ----------------
alias        BOOTIF:0             10.141.240.1    internalnet
physical     BOOTIF [prov]        10.141.0.1      internalnet
Type         Network device name  IP              Network
------------ -------------------- --------------- ----------------
alias        BOOTIF:0             10.141.240.1    internalnet
physical     BOOTIF [prov]        10.141.0.2      internalnet
```

Optionally, each alias node interface can conveniently be assigned a common arbitrary additional host name, perhaps associated with the server, which is CUPS. This does not result in duplicate names here because only one alias interface is active at a time. Setting different additional hostnames for the alias interface to be associated with a unique virtual IP address is not recommended.

Example

```
[basecm11->...interfaces*[BOOTIF:0*]]% set additionalhostnames cups
[basecm11->...interfaces*[BOOTIF:0*]]% commit
```

The preceding can also simply be included as part of the set commands in the foreach statement earlier when the interface was created.

The nodes in the failover group should then be rebooted.

Only the virtual IP address should be used to access the service when using it as a service. Other IP addresses may be used to access the nodes that are in the failover group for other purposes, such as monitoring or direct access.

Service Configuration Adjustments

A service typically needs to have some modifications in its configuration done to serve the needs of the cluster.

CUPS uses port 631 for its service and by default it is only accessible to the local host. Its default configuration is modified by changing some directives within the `cupsd.conf` file. For example, some of the lines in the default file may be:

```
# Only listen for connections from the local machine.
Listen localhost:631
...
# Show shared printers on the local network.
...
BrowseLocalProtocols

...
<Location />
  # Restrict access to the server...
  Order allow,deny

</Location>
...
```

Corresponding lines in a modified `cupsd.conf` file that accepts printing from hosts on the internal network could be modified and end up looking like:

```
# Allow remote access
Port 631
...
# Enable printer sharing and shared printers.
...
BrowseAddress @LOCAL
BrowseLocalProtocols CUPS dnssd
...
<Location />
  # Allow shared printing...
  Order allow,deny
  Allow from 10.141.0.0/16
</Location>
...
```

The operating system that ends up on the failover group nodes should have the relevant service modifications running on those nodes after these nodes are up. In general, the required service modifications could be done:

- with an `initialize` or `finalize` script, as suggested for minor modifications in section 3.20.4
- by saving and using a new image with the modifications, as suggested for greater modifications in section 3.20.2, page 219.

Testing Regular Node HA

To test that the regular node HA works, the active node can have a simulated crash carried out on it like in section 15.2.4.

Example

```
ssh node001
echo c > /proc/sysrq-trigger
~.
```

A passive node then takes over.

15.5.4 The Sequence Of Events When Making Another HA Regular Node Active

The active head node tries to initiate the actions in the following sequence, after the `makeactive` command is run (page 831):

Sequence Of Events In Making Another HA Regular Node Active	
All	Run pre-failover script
Active	Stop service
	Run umount script (stop and show `exit>0` on error)
	Stop active IP address
	Start passive IP address
	Start services on passive.
	Active is now passive
Passive	Stop service
	Run mount script
	Stop passive IP address.
	Start active IP address.
	Start service.
	Passive is now active.
All	Post-failover script

The actions are logged by CMDaemon.
The following conditions hold for the sequence of actions:

- The remaining actions are skipped if the active `umount` script fails.
- The sequence of events on the initial active node is aborted if a STONITH instruction powers it off.
- The actions for `All` nodes is done for that particular failover group, for all nodes in that group.

15.6 HA And Workload Manager Jobs

Workload manager jobs continue to run through a failover handover if conditions allow it.
The 3 conditions that must be satisfied are:

1. The workload manager setup must have been carried out
   (a) during initial installation
   or
   (b) during a run of `cm-wlm-setup`
2. The HA storage setup must support the possibility of job continuity for that workload manager. This support is possible for NAS for Slurm and LSF.

3. Jobs must also not fail due to the shared filesystem being inaccessible during the short period that it is unavailable during failover. This usually depends on the code in the job itself, rather than the workload manager, since workload manager clients by default have timeouts longer than the dead time during failover.

   Already-submitted jobs that are not yet running continue as they are, and run when the resources become available after the failover handover is complete, unless they fail as part of the BCM prejob health check configuration.

16

The Jupyter Notebook Environment Integration

16.1 Introduction

This chapter covers the installation and usage of the Jupyter environment in BCM.

An updated list of the supported Linux distributions and Jupyter functionalities can be found in the feature matrix at `https://service.bcm.nvidia.com/feature-matrix/`, under the `Feature` column, in the section for Jupyter features.

An overview of the concepts and terminology follows.

What Is Jupyter Notebook?

Jupyter Notebook (`https://jupyter-notebook.readthedocs.io/`), or Jupyter, is a client-server open-source application that provides a convenient way for a cluster user to write and execute *notebook documents* in an interactive environment.

In Jupyter, a notebook document, or notebook, is content that can be managed by the application. Notebooks are organized in units called *cells* and can contain both executable code, as well as items that are not meant for execution.

Items not meant for execution can be, for example: explanatory text, figures, formulas, or tables. Notebooks can also store the inputs and outputs of an interactive session.

Notebooks can thus serve as a complete record of a user session, interleaving code with rich representations of resulting objects.

These documents are encoded as JSON files and saved with the `.ipynb` extension. Since JSON is a plain text format, notebooks can be version-controlled, shared with other users and exported to other formats, such as HTML, LaTeX, PDF, and slide shows.

What Is A Notebook Kernel?

A *notebook kernel* (often shortened to *kernel*) is a computational engine that handles the various types of requests in a notebook (e.g. code execution, code completions, inspection) and provides replies to the user (`https://jupyter.readthedocs.io/en/latest/projects/kernels.html`). Usually kernels only allow execution of a single language. There are kernels available for many languages, of varying quality and features.

What Is JupyterHub?

Jupyter on its own provides a single user service. *JupyterHub* (`https://jupyterhub.readthedocs.io/`) allows Jupyter to provide a multi-user service, and is therefore commonly installed with it. JupyterHub is an open-source project that supports a number of authentication protocols, and can be configured in order to provide access to a subset of users.

What Is JupyterLab?

JupyterLab (`https://jupyterlab.readthedocs.io/`) is a modern and powerful interface for Jupyter. It enables users to work with notebooks and other applications, such as terminals or file browsers. It is open-source, flexible, integrated, and extensible.

JupyterLab works out of the box with JupyterHub. It can be used to arrange the user interface to support a wide range of workflows in data science, scientific computing, and machine learning.

JupyterLab is extensible with plugins that can customize or enhance any part of the interface. Plugins exist for themes, file editors, keyboard shortcuts, as well as for other components.

What Is A Jupyter Extension?

Several components of the Jupyter environment can be customized in different ways with extensions. Some types of extensions are:

- IPython extensions (`https://ipython.readthedocs.io/en/stable/config/extensions/#ipython-extensions`)
- Jupyter Notebook server extensions (`https://jupyter-notebook.readthedocs.io/en/stable/extending/index.html`)
- JupyterLab extensions (`https://jupyterlab.readthedocs.io/en/stable/user/extensions.html`)

Extensions are usually developed, bundled, released, installed, and enabled in different ways.

Each extension provides a new functionality for a specific component. For example, JupyterLab extensions can customize or enhance any part of the JupyterLab user interface. Extensions can provide new themes, file viewers, editors and renderers for rich output in notebooks. They can also add settings, add keyboard shortcuts, or add items to the menu or command palette.

What Is Jupyter Kernel Provisionning?

By default, Jupyter runs kernels locally, which can exhaust server resources. A resource manager, such as a workload manager (Slurm, LSF) or Kubernetes, can be used to deal with this issue.

Jupyter Kernel Provisioning (`https://jupyter-client.readthedocs.io/en/latest/provisioning.html#kernel-provisioning`) provides a pluggable interface to distribute kernels across the compute cluster, and uses local underlying resource managers.

The Jupyter Kernel Provisioning framework provides scalability, an improved multi-user support, and a more granular security for Jupyter, in comparison with Jupyter Enterprise Gateway.

In BCM, all the technologies mentioned in these sections are combined to provide a powerful, customizable and user-friendly JupyterLab web interface running on a lightweight, multi-tenant, multi-language, scalable and secure environment, ready for a wide range of enterprise scenarios.

For convenience, in the following sections, *Jupyter* is generally used to collectively refer to Jupyter Notebook, JupyterHub and JupyterLab

16.2 Jupyter Environment Installation

BCM distributes Jupyter via two packages: `cm-jupyter` and `cm-jupyter-local`. They are typically installed via the `cm-jupyter-setup` script (section 16.2.1). Additional packages installed by `cm-jupyter-setup` are `cm-jupyter-eg-kernel-wlm-py312` (used with WLMs), and the optional `cm-jupyter-vnc-local` metapackage (installs distribution-specific VNC packages).

`cm-jupyter` is installed in the `/cm/shared` directory, which is by default exported over NFS. As a result, Jupyter kernels can run on all the compute nodes, without a separate installation to those nodes. `cm-jupyter` provides JupyterHub, JupyterLab, and some extensions.

`cm-jupyter-local` provides the JupyterHub system service (`cm-jupyterhub.service`), and is therefore designed to be installed only on the node exposing users to the web login page for Jupyter. For

convenience, this node is called the *login node*. A login node is typically the head node, but any cluster node can be used.

Since compute nodes are not reachable via a web interface by default, it is the responsibility of the cluster administrator to configure access to these nodes if they are configured to be login nodes while Jupyter runs. That is, login nodes that are compute nodes must have their access configured by assigning IP addresses, configuring the firewall, opening Jupyter ports, and so on. However, if the Jupyter login node is the head node, then BCM takes care of configuring the firewall to open the required ports and of ensuring that the resulting environment is working out of the box.

BCM Jupyter Extensions

For a default deployment of Jupyter, BCM installs and enables the following extensions to the Jupyter environment:

- Jupyter Addons: A Jupyter Notebook server extension that performs API calls to CMDaemon and manages other server extensions;
- Jupyter Kernel Provisioning modules: A set of modules created to handle Jupyter kernels' lifecycles in different possible BCM configurations. The modules available are: Slurm, LSF, Kubernetes.
- Jupyter Kernel Creator (section 16.5): A Jupyter Notebook server extension that provides a new interactive and user-friendly way to create kernels;
- Jupyter VNC (section 16.7): A Jupyter Notebook server extension that enables remote desktops with VNC from notebooks;
- JupyterLab Tools: A JupyterLab extension that exposes BCM server extensions functionalities to the users and shows the Cluster View section;
- Jupyter WLM Magic (section 16.8): An IPython extension that simplifies scheduling of workload manager jobs from the notebook;
- Jupyter Kubernetes Operators Manager (section 16.9): An extension that integrates with Kubernetes clusters, and for which it provides basic overview and management features.

16.2.1 Jupyter Setup

The `cm-jupyter-setup` script can be run on the head node of the cluster to deploy a working Jupyter environment with minimal effort. The script comes with BCM's `cm-setup` package. It has no prerequisites, and can be run before or after configuring any resource manager, such as Kubernetes or Slurm.

In the final summary screen, as with other scripts that come with the `cm-setup` package, the configuration setting can be one of the following kinds:

- `minimal config`: For this minimal kind of configuration setting, the configuration consists only of the settings that differ from the default application configuration. The remaining default application configuration settings are not included in the configuration.
- `config`: For this non-minimal kind of configuration setting, the configuration consists of the minimal configuration settings (the ones changed by BCM), as well as the remaining default configuration settings (the ones provided with the application). That is to say: the remaining default configuration application settings are included in the configuration. The default application configuration settings may change as the upstream packages change.

By default, the Jupyter environment initially contains only Jupyter's default Python 3 kernel, which runs on the login node.

During setup, an administrator can deploy the Jupyter login interface on multiple nodes to evenly distribute the load across them. In this case the administrator must configure a load balancer to route users' requests across those nodes.

These login nodes become members of the same `configurationoverlay`, and therefore share the same Jupyter configuration, such as port numbers, authenticator, and so on.

By default, the Jupyter configuration file points to local SSL certificates. This means that if there are multiple Jupyter login interfaces, then each node uses its own SSL certificate.

16.2.2 Jupyter Architecture

The default Jupyter architecture deployed by `cm-jupyter-setup` is shown in figure 16.1.

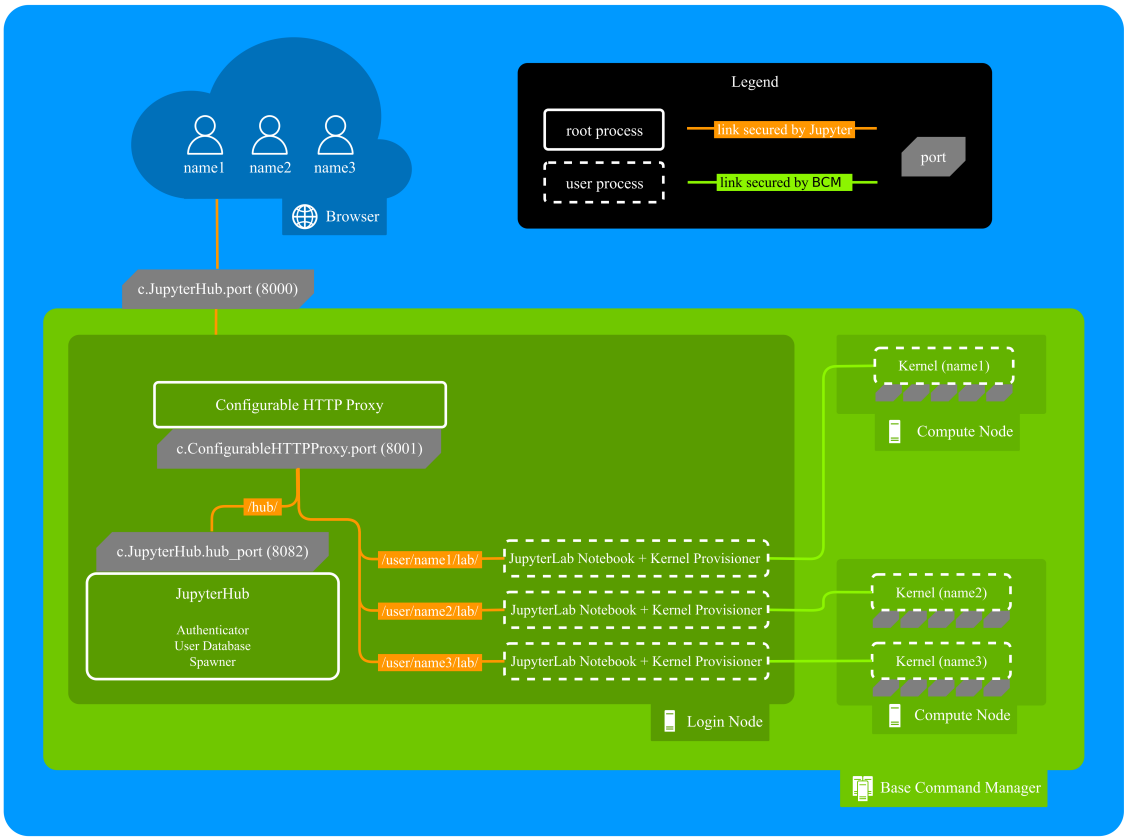


Figure 16.1: Jupyter architecture

In this architecture, the `cm-jupyterhub.service` provided by `cm-jupyter-local` starts JupyterHub on the port specified by `c.JupyterHub.hub_port` (default: 8901) on a login node (typically the head node).

JupyterHub then automatically spawns a dynamic proxy to route HTTP requests via BCM's `cm-npm-configurable-http-proxy` package.

The proxy is the only process that listens for clients' requests on Jupyter's public interface, as specified by `c.JupyterHub.port` (default: 8000).

JupyterHub instructs the proxy on how requests should be dispatched by using its REST API. This is exposed at the port specified by `c.ConfigurableHTTPProxy.port` (default: 8902), typically defined within `c.ConfigurableHTTPProxy.api_url`.

At this stage, users accessing Jupyter with their browsers are either redirected to the shared Hub (e.g. for authentication), or to their dedicated single-user servers to run notebooks.

By default, single-user servers are accessed at `/user/<`*username*`>`, while JupyterHub is accessed at

/hub (https://jupyterhub.readthedocs.io/en/stable/reference/technical-overview.html).

The cm-jupyter-setup script automatically installs JupyterLab and sets /lab as the default URL (c.Spawner.default_url) to redirect users to the new interface.

Finally, JupyterHub is integrated to spawn the JupyterLab interface on a login node by configuring jupyterhub.spawner.LocalProcessSpawner as the default spawning mechanism (c.JupyterHub.spawner_class), and jupyterhub-singleuser-gw as the default spawning command (c.Spawner.cmd).

When a new notebook is started, a Jupyter kernel provisioner scans for an available port, and spawns a kernel chosen by the user on an appropriate cluster node. This process is then connected to JupyterLab.

JupyterHub and its HTTP proxy are run as two root processes, while JupyterLab and the in-built kernel provisioning run as user processes.

The privileges of the kernels spawned by Jupyter Kernel Provisioning can be configured by cluster administrators, and depend on the underlying computational engine (for example, Kubernetes). By default, all the kernels configurable by BCM are run as user processes, and no privilege escalation is possible.

Communication between users' browsers, JupyterHub, its HTTP proxy, and JupyterLab, is secured by default by JupyterHub. On the other hand, communication between JupyterLab and the kernels on the nodes, is secured by BCM.

Administrators can customize their Jupyter integration by setting some of the aforementioned options when the cm-jupyter-setup script runs. New values are automatically handled by BCM and written to Jupyter configuration files.

Other configuration options can be found in /cm/local/apps/jupyter/current/conf/jupyterhub_config.py.

16.2.3 Verifying Jupyter Installation

The cm-jupyter-setup script automatically starts the cm-jupyterhub service.

Any user (not necessarily root) can then verify the installation is working as expected. Here an ordinary user, jupyterhubuser runs the checks.

It can take some time until the service is fully up and running, even if a status check with systemctl shows that the service is active:

Example

```
[jupyterhubuser@basecm11 ~]$ systemctl status cm-jupyterhub -l
  cm-jupyterhub.service - JupyterHub
     Loaded: loaded (/lib/systemd/system/cm-jupyterhub.service; static)
     Active: active (running) since Mon 2025-01-06 14:47:28 CET; 4h 25min ago
   Main PID: 4367 (run.sh)
      Tasks: 11 (limit: 19051)
     Memory: 190.2M
        CPU: 1min 46.193s
     CGroup: /system.slice/cm-jupyterhub.service
             |-4367 /bin/bash /cm/shared/apps/jupyter/16.0.0/bin/run.sh
             |-4428 /cm/local/apps/python312/bin/python3 /cm/shared/apps/jupyter/16.0.0/bin/jupyterhub ...
             |-4429 tee -a /var/log/jupyterhub.log
             '-5431 node /cm/shared/apps/jupyter/16.0.0/bin/configurable-http-proxy --ip "" --port 8000 ...
```

A check can then be done to see that the Jupyter extensions provided by BCM are installed and enabled:

Example

```
[jupyterhubuser@basecm11 ~]$ module load jupyter
Loading jupyter/16.0.0
```

```
  Loading requirement: python312
[jupyterhubuser@basecm11 ~]$ jupyter labextension list
JupyterLab v4.3.3
/cm/shared/apps/jupyter/current/share/jupyter/labextensions
        jupyterlab_pygments v0.3.0 enabled OK (python, jupyterlab_pygments)
        @brightcomputing/jupyterlab-tools v0.2.7 enabled OK (python, brightcomputing_jupyterlab_tools)
        @jupyter-widgets/jupyterlab-manager v5.0.13 enabled OK (python, jupyterlab_widgets)
        @jupyterhub/jupyter-server-proxy v4.4.0 enabled OK
```

16.2.4 Login Configuration

User Access To JupyterHub With The Default Configuration

Once JupyterHub is functioning, its web login interface is accessible with a browser using the HTTPS protocol on the specified port (figure 16.2):

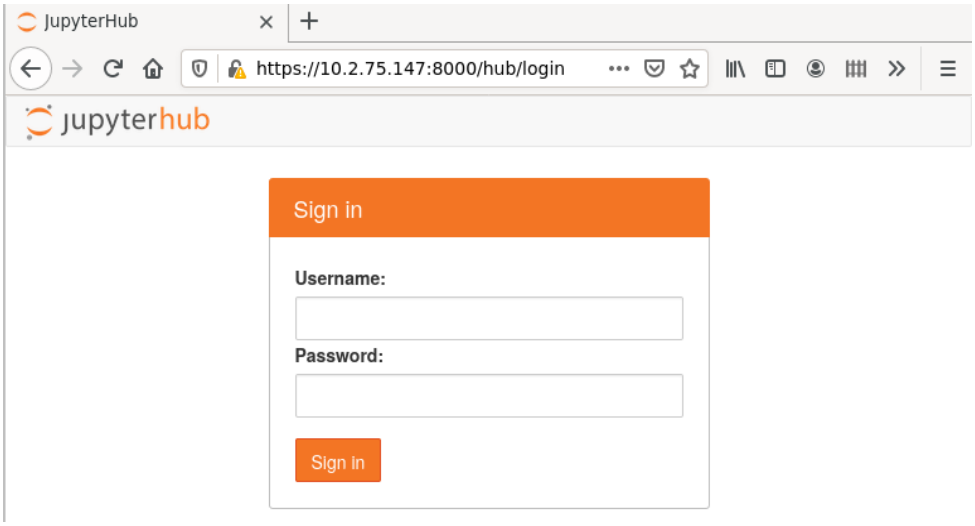


Figure 16.2: JupyterHub login screen

Running as root under JupyterHub is not recommended, so logging in to JupyterHub as root is not allowed by default.

Any other PAM user can log in to JupyterHub.

If needed, a test user `jupyterhubuser` with password `jupyterhubuser` can be created with, for example:

Example

```
[root@basecm11 ~]# cmsh
[basecm11]% user
[basecm11->user]% add jupyterhubuser
[basecm11->user*[jupyterhubuser*]]% set password
enter new password:
retype new password:
[basecm11->user*[jupyterhubuser*]]% commit
```

Restricting User Access To JupyterHub

JupyterHub logins can be limited to members of particular groups.

For example, `alice` could be made a member of the group `jupyterusers`:

Example

```
[root@basecm11 ~]# cmsh
[basecm11]% group
[basecm11->group]% add jupyterusers
[basecm11->group]*[jupyterusers*]% append members alice
[basecm11->group]*[jupyterusers*]% commit
```

The group jupyterusers could then be set to be allowed to authenticate to JupyterHub with an authentication policy:

Example

```
[root@basecm11 ~]# cmsh
[basecm11]% configurationoverlay
[basecm11->configurationoverlay]% use jupyterhub
[basecm11->configurationoverlay[jupyterhub]]% roles
[basecm11->configurationoverlay[jupyterhub]->roles]% use jupyterhub
[basecm11->configurationoverlay[jupyterhub]->roles[jupyterhub]]% configs
[basecm11->...roles[jupyterhub]->configs]% add c.BrightAuthenticator.groups_allow
[basecm11->...roles*[jupyterhub*]->configs*[c.BrightAuthenticator.groups_allow*]]% set value [\"jupyterusers\"]
[basecm11->...roles*[jupyterhub*]->configs*[c.BrightAuthenticator.groups_allow*]]% commit
```

Similarly, restricting authentication to JupyterHub to a specific group of users can be done by configuring c.BrightAuthenticator.groups_deny.

Another way to restrict user access is based on the CMDaemon profile set for the user. In that case the JupyterHub configuration settings are c.BrightAuthenticator.cmd_profiles_allow and c.BrightAuthenticator.cmd_profiles_deny.

In the following example, users with a readonly profile are not allowed to authenticate for JupyterHub.

Example

```
[basecm11->...roles[jupyterhub]->configs[c.BrightAuthenticator.groups_allow]]% ..
[basecm11->...roles[jupyterhub]->configs]% add c.BrightAuthenticator.cmd_profiles_deny
[basecm11->...roles*[jupyterhub*]->configs*[c.BrightAuthenticator.cmd_profiles_deny*]]% set value [\"readonly\"]
[basecm11->...roles*[jupyterhub*]->configs*[c.BrightAuthenticator.cmd_profiles_deny*]]% commit
```

A table summarizing the authentication policies is:

Key	Value
c.BrightAuthenticator.groups_allow	users in the specified groups can authenticate
c.BrightAuthenticator.groups_deny	users in the specified groups cannot authenticate
c.BrightAuthenticator.cmd_profiles_allow	users with the specified profiles can authenticate
c.BrightAuthenticator.cmd_profiles_deny	users with the specified profiles cannot authenticate

16.2.5 JupyterHub Screen After Login

After the first login, a new single-user server is spawned (figure 16.3):

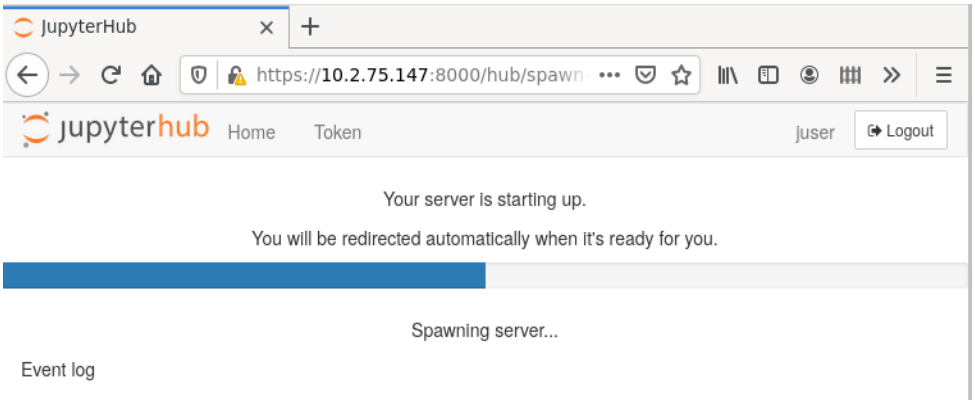


Figure 16.3: JupyterHub starting single-user server

Users are redirected to the JupyterLab interface and have access to Jupyter's default Python 3 kernel (figure 16.4):

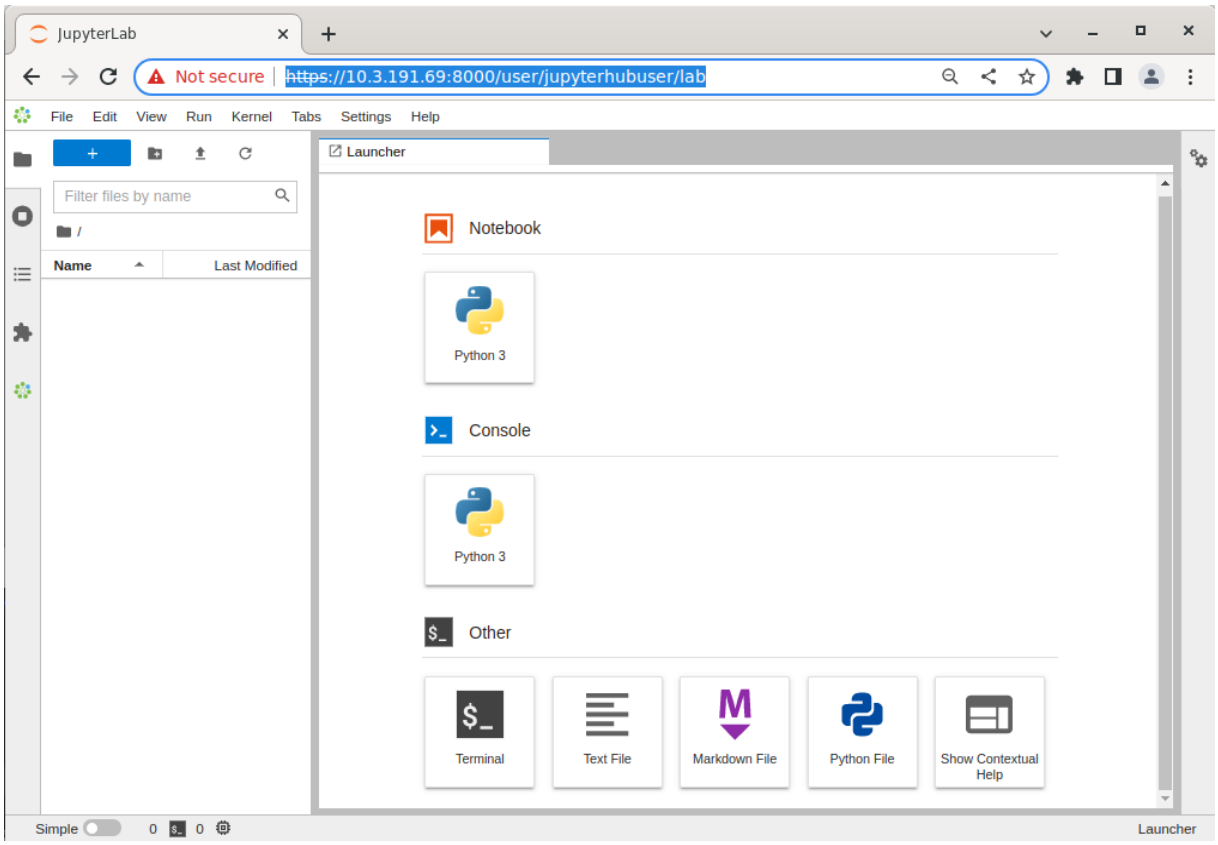


Figure 16.4: JupyterLab Launcher

If using Kubernetes under Jupyter, then a user registered under the Linux-PAM system must be added separately via Kubernetes with `cm-kubernetes-setup`. (section 4.11 of the *Containerization Manual*).

16.3 Jupyter Notebook Examples

The `cm-jupyter` package (section 16.2.1) provides a number of machine learning notebook examples that can be executed with Jupyter.

The notebooks include some applications developed with TensorFlow, PyTorch, MXNet, and other frameworks. The applications can be found in the /cm/shared/examples/jupyter/notebooks/ directory:

```
[jupyterhubuser@basecm11 ~]$ ls /cm/shared/examples/jupyter/notebooks/
Keras+TensorFlow2-addition.ipynb  psql-example.ipynb      Spark+XGBoost-mortgage.ipynb
llm-bcm-manuals-rag               Pytorch-cartpole.ipynb  TensorFlow-minigo.ipynb
llm-codellama-local               R-iris.ipynb
MXNet-superresolution.ipynb       Spark-pipeline.ipynb
```

The datasets needed to execute these notebooks can be found in the /cm/shared/examples/jupyter/datasets/ directory:

```
[jupyterhubuser@basecm11 ~]$ ls /cm/shared/examples/jupyter/datasets/
880f8b8a6fd-mortgage-small.tar.gz  kaggle-iris.csv
```

Users can copy these examples to their home directories, create or choose appropriate kernels to execute them, and interactively run them from Jupyter. In order to edit notebooks, the write permissions must be kept during the copy.

It is the responsibility of users to make sure that the required modules are loaded by their Jupyter kernels. The list of frameworks and libraries required to run an example is usually available at the beginning of each notebook.

16.4 Jupyter Kernels

In Jupyter, kernels are defined as JSON files.

Any user that the cluster administrator has registered in the Linux-PAM system can list the available Jupyter kernels via the command line. The following example is run in the initial Jupyter environment:

Example

```
[jupyterhubuser@basecm11 ~]$ module load jupyter
[jupyterhubuser@basecm11 ~]$ jupyter kernelspec list
Available kernels:
  python3    /cm/shared/apps/jupyter/current/share/jupyter/kernels/python3
```

Each kernel directory contains a kernel.json file describing how Jupyter spawns that kernel:

Example

```
[jupyterhubuser@basecm11 ~]$ ls /cm/shared/apps/jupyter/current/share/jupyter/kernels/*/kernel.json
/cm/shared/apps/jupyter/current/share/jupyter/kernels/python3/kernel.json
```

In addition to specifications for shared kernels, each user can define new personal ones in the home directory. By default, the Jupyter data directory for a user is located at $HOME/.local/share/jupyter.

This path can be verified with Jupyter by using the --paths option:

Example

```
[jupyterhubuser@basecm11 ~]$ jupyter --paths
config:
    /home/jupyterhubuser/.jupyter
    /home/jupyterhubuser/.local/etc/jupyter
    /cm/local/apps/jupyter/conf
    /cm/shared/apps/jupyter/current/etc/jupyter
data:
    /home/jupyterhubuser/.local/share/jupyter
    /cm/shared/apps/jupyter/current/share/jupyter
runtime:
    /home/jupyterhubuser/.local/share/jupyter/runtime
```

The simplest definition for a Python3 kernel designed to run on the login node is:

```
{
 "argv": ["python",
    "-m",
    "ipykernel_launcher",
    "-f",
    "{connection_file}"
 ],
 "display_name": "Python 3",
 "language": "python"
}
```

In the preceding kernel definition:

- `argv`: is the command to be executed to locally spawn the kernel
- `"display_name"`: is the name to be displayed in the JupyterLab interface
- `"language"`: is the supported programming language (`"language"`)
- `"{connection_file}"` (`https://jupyter-client.readthedocs.io/en/stable/kernels.html#connection-files`) is a placeholder, and is replaced by Jupyter with the actual path to the connection file before starting the kernel.

The following kernel is Jupyter's default Python 3 kernel distributed by BCM in the initial environment:

Example

```
[jupyterhubuser@basecm11 ~]$ cat /cm/shared/apps/jupyter/current/share/jupyter/kernels/python3/kernel.json

{
 "argv": [
  "/cm/local/apps/python312/bin/python3.12",
  "-m",
  "ipykernel_launcher",
  "--InteractiveShellApp.extra_extensions=cm_jupyter_wlm_magic",
  "--TerminalIPythonApp.extra_extensions=cm_jupyter_wlm_magic",
  "-f",
  "{connection_file}"
 ],
 "display_name": "Python 3",
 "language": "python",
 "env": {
  "PYTHONPATH": "/cm/shared/apps/jupyter/current/lib64/python3.12/site-packages:/cm/shared/apps/jupyter/
current/lib/python3.12/site-packages"
 }
}
```

The two kernels are not very different. They differ from each other in the Python 3 binary path, the IPython extension (Jupyter WLM Magic), and the exported `PYTHONPATH` environment variable (`"env"`).

16.4.1 Jupyter Kernel Provisioning Kernels

Jupyter is designed to run both the kernel processes, as well as the user interface (JupyterLab or Jupyter Notebook) on the same host. The kernel `{connection_file}` is therefore stored in the `~/.local/share/jupyter/runtime` directory, or in the `/run` directory.

JupyterLab can delegate the task of spawning kernels to another component, Jupyter Kernel Provisioning, as defined in the kernel's JSON file for `kernel_provisioner`. Jupyter Kernel Provisioning allows the complete life-cycle of several Jupyter kernels to be managed at the same time—their start, status monitoring, and termination, but the particular Jupyter kernels that run are otherwise independent of Jupyter Kernel Provisioning, and can be managed by third parties.

Jupyter Kernel Provisioning requires an extended `kernel.json` definition to describe a particular process-proxy module to handle the kernel.

A simple definition for a Python3 kernel designed to be scheduled via JEG is:

```
{
  "display_name": "Python 3.12 via SLURM 250102185019",
  "language": "python",
  "metadata": {
    "kernel_provisioner": {
      "provisioner_name": "slurm-provisioner",
      "config": {
        "timeout": 60,
        "response_manager": {
            "version": 2
        },
        "submit_cmd": {
            "path": "templates/submit_cmd.sh.j2",
            "vars": {
                "modules": "shared slurm jupyter-eg-kernel-wlm-py312"
            }
        },
        "query_cmd": {
            "path": "templates/query_cmd.sh.j2",
            "vars": {
                "modules": "shared slurm jupyter-eg-kernel-wlm-py312"
            }
        },
        "info_cmd": {
            "path": "templates/info_cmd.sh.j2",
            "vars": {
                "modules": "shared slurm jupyter-eg-kernel-wlm-py312"
            }
        },
        "cancel_cmd": {
            "path": "templates/cancel_cmd.sh.j2",
            "vars": {
                "modules": "shared slurm jupyter-eg-kernel-wlm-py312"
            }
        },
        "submit_script": {
            "path": "templates/submit_script.sh.j2",
            "vars": {
                "job_prefix": "jupyter-kernel-slurm-py312",
                "partition": "",
                "ntasks": "1",
                "gres": "",
```

```
                "work_dir": "/home/alice",
                "modules": "shared slurm jupyter-eg-kernel-wlm-py312",
                "oversubscribe": false,
                "pythonuserbase_loc": "temp"
            }
        }
      }
    }
  },
  "argv": []
}
```

In this example, the "metadata" entry has been added. It includes "kernel_provisioner" and "provisioner_name", which define the exact provisioner being used to manage the life-cycle of the kernel. The provisioner is defined via Python's entry points specification.

It also contains paths to several script templates used to spawn the job within the context that the kernel process runs, and sets the corresponding environment variables and other variables for the templates. The "argv": [] is empty because it is replaced by a script defined in templates/submit_script.sh.j2

BCM is equipped with several types of provisioners to interact with a wide range of resource managers, such as Kubernetes or Slurm. This allows kernels to be scheduled across compute nodes.

BCM recommends that kernels using the Jupyter Kernel Provisioning mechanism are created and used with the Jupyter Kernel Creator (section 16.5) extension.

16.4.2 Tunables For Kernel Provisioners

BCM provides defaults for all the templates. The aim is to have the kernels that are created just work on a typical cluster. However a better fit to the running environment may be possible with some further fine-tuning.

Configuration parameters for modifying the kernel templates can be added with cmsh. These parameters are put into the Jupyter configuration file at /cm/local/apps/jupyter/conf/jupyterhub_config.py, and become accessible when JupyterLab starts. Templates or already-created kernels can be edited—which might be a preferred approach to test a parameter before simply adding it via cmsh. Parameters that are set in the kernel specification (the kernel.json file) have precedence over the ones set in jupyterhub_config.py

The following example shows a session that adds the c.KernelResponseManager.public_ip configuration parameters within cmsh:

Example

```
[root@basecm11 ~]# cmsh
[basecm11]% configurationoverlay
[basecm11->configurationoverlay]% use jupyterhub
[basecm11->configurationoverlay[jupyterhub]]% roles
[basecm11->configurationoverlay[jupyterhub]->roles]% use jupyterhub
[basecm11->configurationoverlay[jupyterhub]->roles[jupyterhub]]% configs
[basecm11->...]->roles[jupyterhub]->configs]% add c.KernelResponseManager.public_ip
[basecm11->...]->roles*[jupyterhub*]->configs*[c.KernelResponseManager.public_ip*]]% set value "'10.10.1.1'"
[basecm11->...]->roles*[jupyterhub*]->configs*[c.KernelResponseManager.public_ip*]]% commit
```

On commit, the cm-jupyter service is restarted, which means that all user sessions are dropped. To avoid this, editing the templates directly in the /cm/shared/apps/jupyter/current/share/jupyter/kerneltemplates directory can be considered.

Table 16.4.2: Jupyter Kernel Tunables

Configuration parameter	Path in kernel.json (metadata.kernel_provisioner.config)	Default	Description
c.CMKernelProvisionerBase.timeout	.timeout	5	Timeout starting kernel
c.CMKernelProvisionerBase.include_regex_env	.include_regex_env	^(.+_API_(KEY|TOKEN|HOST|TYPE|ORG_ID|ENDPOINT)|HF_.+)$	Regex for environment variable names to be inherited from JupyterLab process running on login node
c.CMKernelProvisionerBase.exclude_regex_env	.exclude_regex_env	^(JPY_API_TOKEN|JUPYTERHUB_.+|PYTHON.+|JUPYTERLAB_.+|PATH|LD_LIBRARY_PATH.*)$	Regex for environment variable names not to be passed to running kernels
c.CMJKProvisioner.poll_interval	.poll_interval	5	Polling interval and interval between retries for k8s operations
c.CMJKProvisioner.operation_timeout	.operation_timeout	5	Timeout for running commands interacting with k8s
c.KernelResponseManager.public_ip	.response_manager.public_ip	Detected automatically	The IP address on the login node that jupyter-kernel-starter will use for callbacks when the kernel is started on the compute node
c.KernelResponseManager.public_network	.response_manager.public_network	Detected automatically	The network on the login node that jupyter-kernel-starter will use for callbacks when the kernel is started on the compute node
c.KernelResponseManager.public_hostname	.response_manager.public_hostname	Detected automatically	External hostname of the login node can be specified, and the public IP address will be detected by resolving
c.KernelResponseManager.bind_ip	.response_manager.bind_ip	Detected automatically	The IP address used by the response manager to bind the socket that listens for callbacks from jupyter-kernel-starter

...continues

...continued

Configuration parameter	Path in kernel.json (metadata.kernel_provisioner.config)	Default	Description
c.KernelResponseManager.bind_network	.response_manager.bind_network	Detected automatically	The IP address used by the response manager to bind the socket that listens for callbacks from jupyter-kernel-starter
c.KernelResponseManager.bind_port_range_start	.response_manager.bind_port_range_start	1025	The start of the port range within which the response manager tries to obtain a port, for listening to callbacks from the kernel
c.KernelResponseManager.bind_port_range_end	.response_manager.bind_port_range_end	65535	The end of the port range within which the response manager tries to obtain a port, for listening to callbacks from the kernel
c.KernelResponseManager.bind_port_retries	.response_manager.bind_port_retries	16	How many times the response manager tries to find a free port
c.KernelResponseManager.max_in_requests	.response_manager.max_in_requests	5	How many incoming messages to the kernel starter can be buffered
c.KernelResponseManager.max_out_requests	.response_manager.max_out_requests	5	How many outgoing messages to the kernel starter can be buffered

The *response manager* in the preceding table is a part of WLM kernel provisioners. It is dedicated to getting callbacks and managing signal communications with the running kernel. It acts as a proxy for system signals and informs the kernel provisioner about the kernel being started and which ports it is listening to. The response manager works in tandem with jupyter-kernel-starter.

16.5 Jupyter Kernel Creator Extension

Creating or editing kernels can be cumbersome and error-prone for users, depending on the features of the execution context desired for their notebooks.

To provide a more user-friendly experience, BCM includes the *Jupyter Kernel Creator* extension in JupyterLab. This extension is accessed from the navigation pane in the JupyterLab interface, by clicking on the BCM icon.

Jupyter Kernel Creator allows users to create kernels using the JupyterLab interface, without the need to directly edit JSON files. With this interface users can create kernels by customizing an available

template according to their needs.

A template can be considered to be the skeleton of a kernel, with several preconfigured options, and others options that are yet to be specified. Common customizations for templates include environment modules to be loaded, workload manager queues to be used, number and type of GPUs to acquire, and so on.

Templates are usually defined by administrators according to cluster capabilities, programming languages and user requirements. Each template can provide different options for customizations.

Administrators often create different templates to take advantage of different workload managers, programming languages and hardware resources. For example, an administrator may define a template for scheduling Python kernels via Kubernetes, another one for R kernels via Slurm, and yet another one for Bash kernels via Platform LSF.

16.5.1 BCM Predefined Kernel Templates

To simplify Jupyter configuration for administrators, BCM distributes a number of pre-defined templates with Jupyter Kernel Creator. These templates can be used for default configurations of BCM workload managers, and can be customized and extended for more advanced use. Kernel templates defined by BCM can be found in the Jupyter installation directory, under the `kerneltemplates` directory:

```
[jupyterhubuser@basecm11 ~]$ ls /cm/shared/apps/jupyter/current/share/jupyter/kerneltemplates/
filter.yaml            k8s-cmjkop-py    lsf-py312           slurm-py312   slurm-pyxis-r   k8s-cmjkop-julia
k8s-cmjkop-py-spark    slurm-py-conda   k8s-cmjkop-ngc-py   lsf-bash      slurm-bash      slurm-pyxis-py
```

BCM Predefined Kernel Templates Seen By Users

Users can view the available predefined kernel templates in the Jupyter web browser interface, within the `KERNEL TEMPLATES` section of the dedicated BCM extensions panel (figure 16.5):

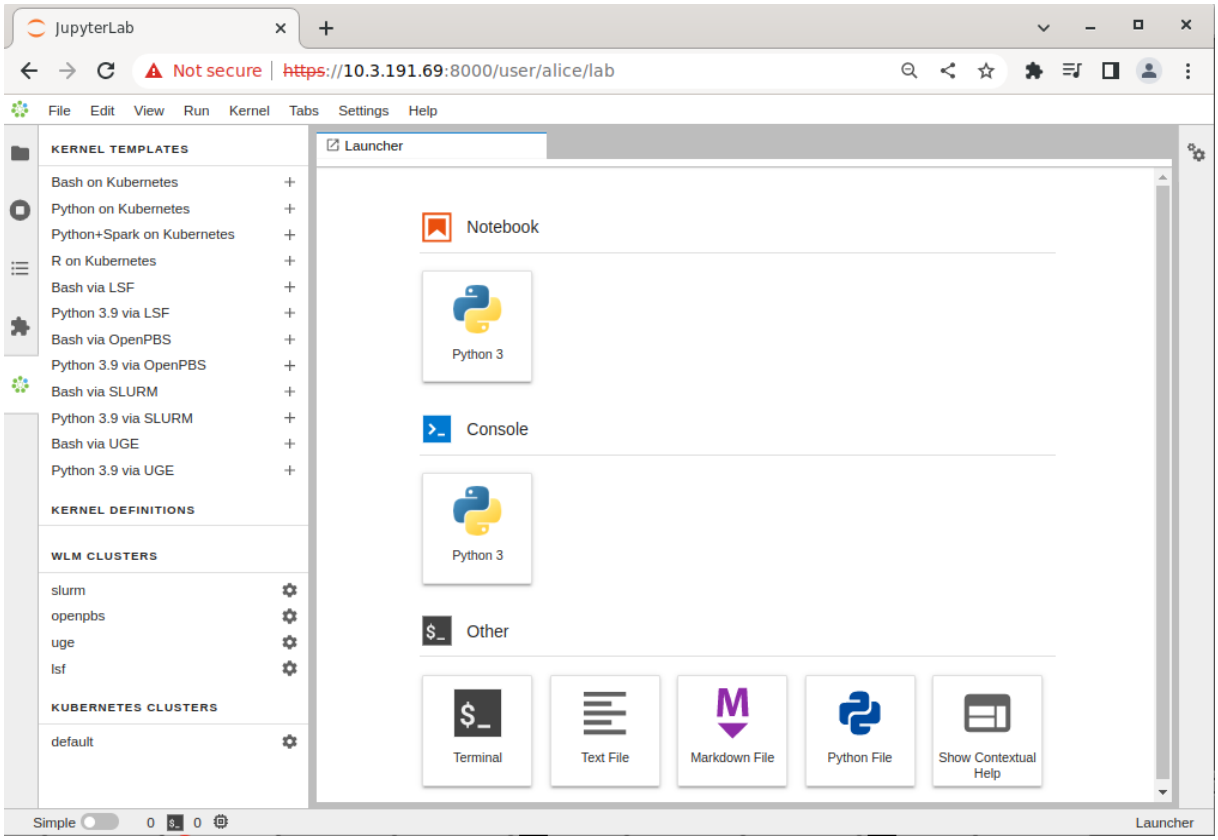


Figure 16.5: JupyterLab BCM extensions section with kernel templates

However, the templates provided by BCM are listed in the panel only if they can be used on the cluster. This means that the templates are listed only after the associated workload manager instance, or associated Kubernetes configuration (such as a Kubernetes operator), have been deployed by cluster manager utilities. For example:

- After running the `cm-kubernetes-setup` cluster manager utility to deploy a Kubernetes cluster, the Kubernetes cluster instance is displayed (navigation path: *menu* > *dedicated BCM extension panel* > *Kubernetes clusters section*)

  Then, after running the `cm-jupyter-kernel-operator` cluster manager utility to deploy a Jupyter kernel operator package, and configuring a user (section 6.6 of the *Containerization Manual*), the templates `k8s-cmjkop-julia`, `k8s-cmjkop-py`, and `k8s-cmjkop-py-spark` become available.

  The templates are listed as:

  - `Julia on Kubernetes Operator`
  - `Python on Kubernete Operator`
  - `Python+Spark on Kubernetes Operator`

  and accessed via the navigation path: *menu* > *dedicated BCM extension panel* > *kernel templates section*.

Users can instantiate a kernel template to create an actual kernel from the dedicated BCM extensions section using the + button of the template. A dialog is dynamically generated for the template being instantiated, and users are asked to fill a number of customization options defined by administrators (figure 16.6):

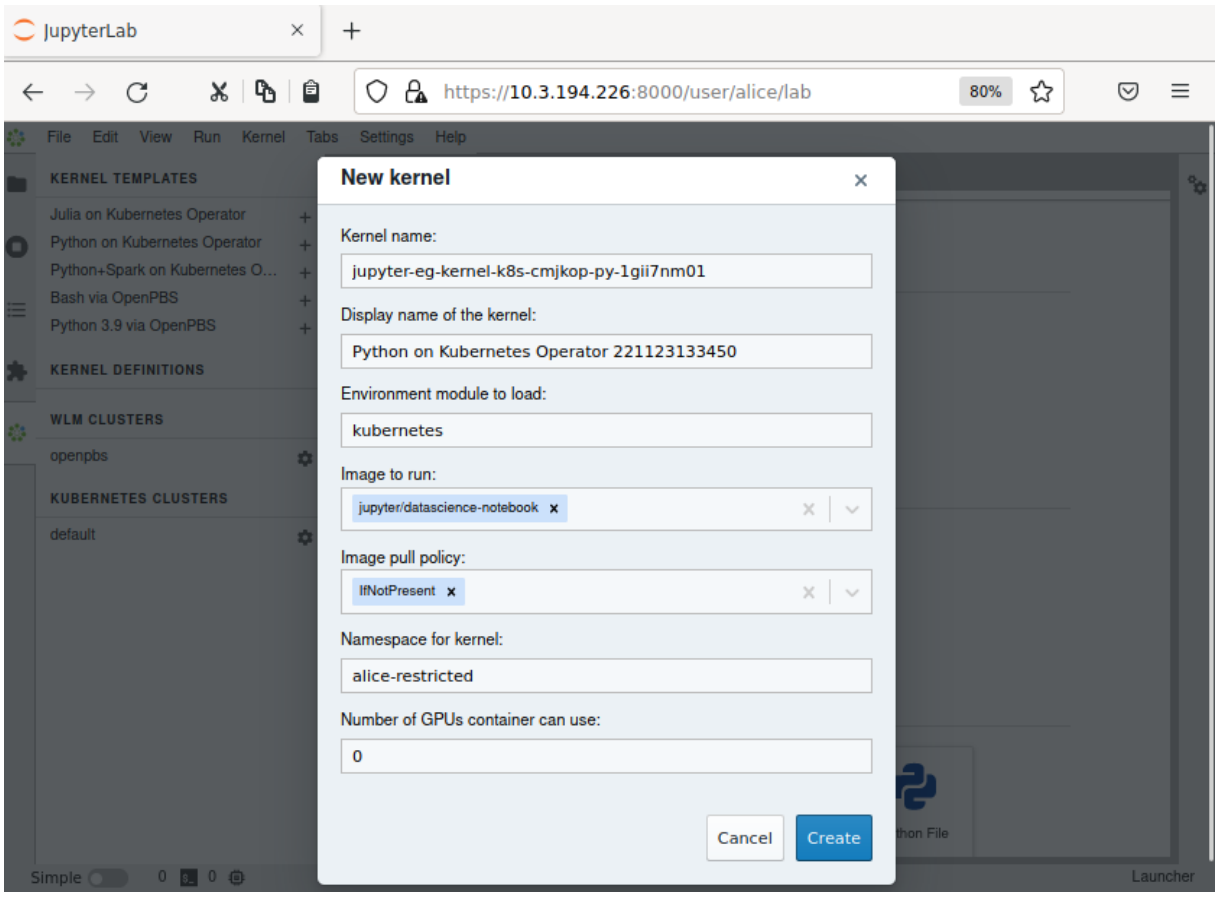


Figure 16.6: Jupyter kernel template customization screen

Once the template is completely customized, the kernel can be created. It automatically appears in the JupyterLab Launcher screen (figure 16.7) and can be used to run notebooks or a console session (figure 16.7):

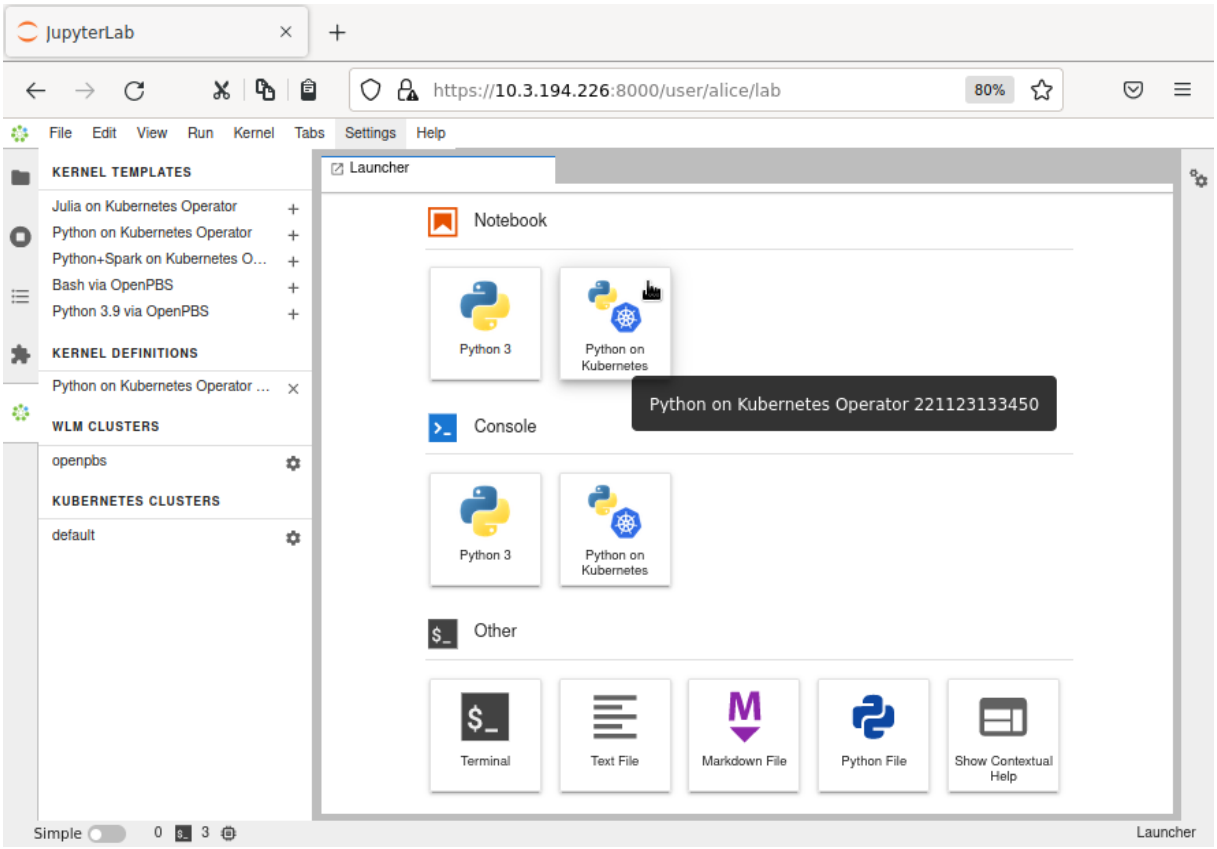


Figure 16.7: JupyterLab Launcher screen with new custom kernel

A user who lists available Jupyter kernels via the command line now sees the newly-created kernel:

```
[jupyterhubuser@basecm11 ~]$ module load jupyter
[jupyterhubuser@basecm11 ~]$ jupyter kernelspec list
Available kernels:
     k8s-cmjkop-py-1gii7nm01    /home/alice/.local/share/jupyter/kernels/k8s-cmjkop-py-1gii7nm01
     python3                    /cm/shared/apps/jupyter/current/share/jupyter/kernels/python3
```

The new kernel directory will contain the JSON definition generated by the Jupyter Kernel Creator:

```
[jupyterhubuser@basecm11 ~]$ cat .local/share/jupyter/kernels/k8s-cmjkop-py-1gii7nm01/kernel.json

{
  "language": "python",
  "display_name": "Datascience Notebook Kernel",
  "metadata": {
    "kernel_provisioner": {
      "provisioner_name": "cmjk-provisioner",
      "config": {
        "timeout": 280,
        "template": {
          "path": "templates/cmjk.yaml.j2",
          "env_module": "kubernetes",
          "vars": {
```

```
            "image": "quay.io/jupyter/datascience-notebook",
            "namespace": "alice-restricted",
            "image_pull_policy": "IfNotPresent",
            "gpu_limit": 0,
            "pythonuserbase_loc": "temp"
          }
        }
      }
    }
  },
  "argv": []
}
```

Jupyter kernel names need not be unique. Users should therefore choose meaningful and distinguishable display names for their kernels. Doing so makes the JupyterLab Launcher screen easier to use.

For convenience, a summary of the available kernel templates and their requirements is shown in table 16.1:

Table 16.1: *Available Jupyter kernel templates for BCM and their requirements*

Template name	Requirement	Description
`k8s-cmjkop-julia`	Kubernetes	Jupyter official image via Jupyter Kernel Operator (using Julia)
`k8s-cmjkop-ngc-py`	Kubernetes	NGC images via Jupyter Kernel Operator
`k8s-cmjkop-py`	Kubernetes	Jupyter official image via Jupyter Kernel Operator (Python)
`k8s-cmjkop-py-spark`	Kubernetes[1]	Python + Spark via Jupyter Kernel Operator (using Python and Spark)
`lsf-bash`	Platform LSF	Bash via Platform LSF
`lsf-py312`	Platform LSF	Python 3.12 via Platform LSF
`slurm-bash`	Slurm	Bash via Slurm
`slurm-py312`	Slurm	Python 3.12 via Slurm
`slurm-py-conda`	Slurm + Conda[2]	Python 3.12 and Conda via Slurm
`slurm-pyxis-py`	Slurm + Enroot	Python running inside imported Pyxis+Enroot image in Slurm
`slurm-pyxis-r`	Slurm + Enroot	R running inside imported Pyxis+Enroot image in Slurm

...continues

Table 16.1: Available Jupyter kernel templates...continued

Template name	Requirement	Description

[1] Docker image: *docker.io/brightcomputing/jupyter-kernel-sample:k8s-spark-3.5.3-py38-cuda12.6-rapids24.08-2*
[2] Conda needs to be installed for the user, and the Conda environment needs to be configured in the user's Bash shell as described in section 10.4.2 of the *User Manual*.
DockerHub kernels page: `https://hub.docker.com/r/brightcomputing/jupyter-kernel-sample/tags`

16.5.2 Jupyter Kernel Starter

Jupyter Kernel Starter, implemented as the software `jupyter-kernel-starter`, runs alongside the Jupyter kernel. It acts as a starter and sidecar to manage the life-cycle of the kernel on the node within the WLM job context. It informs the kernel provisioner, via the response manager, about the kernel status, and which ports are being used by Jupyter Kernel Manager (the kernel provisioner layer manager). It also performs basic heartbeat and watchdog operations to make sure that the running kernel does not drain cluster resources if the connection to the kernel provisioner is lost.

```
[jupyterhubuser@basecm11 ~]$ module load jupyter-eg-kernel-wlm-py312
[jupyterhubuser@basecm11 ~]$ jupyter-kernel-starter --help
usage: jupyter-kernel-starter [-h] [--log-level <log_level>] [--kernel-id <kernel_id>]
                              [--response-address <ip>:<port>] [--bind-ip <x.x.x.x>]
                              [--bind-network <x.x.x.x/y>] [--port-range <begin>..<end>]
                              [--port-bind-attempts <n>] [--wait-ports-timeout <sec>]
                              [--shutdown-timeout <sec>] [--response-server-socket-timeout <sec>]
                              [--watchdog-interval <sec>] [--kernel-script <cmd>]
                              [--kernel-script-base64 <cmd_base64>] [--connection-file-dir <path>]
                              [--encryption-key <pem_key>]

options:
  -h, --help            show this help message and exit
  --log-level <log_level>
                        Set the logging level (DEBUG, INFO, WARNING, ERROR, CRITICAL). Can be specified as the
                        JKS_LOG_LEVEL environment variable.
  --kernel-id <kernel_id>
                        ID of the Jupyter kernel
  --response-address <ip>:<port>
                        Address of the server to return kernel connection info. Can be specified as the
                        JKS_RESPONSE_ADDRESS environment variable.
  --bind-ip <x.x.x.x>   IP address to bind. Can be specified as the JKS_BIND_IP environment variable.
  --bind-network <x.x.x.x/y>
                        Alternatively IP network to bind. Can be specified as the JKS_BIND_NETWORK environment
                        variable.
  --port-range <begin>..<end>
                        Range of the ports to use for kernel. Can be specified as the JKS_PORT_RANGE environment
                        variable.
  --port-bind-attempts <n>
                        How many times to try to find unoccupied ports to use. Can be specified as the
                        JKS_PORT_BIND_ATTEMPTS environment variable.
  --wait-ports-timeout <sec>
                        How many seconds the kernel waits for ports to open before sending connection information
to the server. Can be specified as the JKS_WAIT_PORTS_TIMEOUT environment variable.
  --shutdown-timeout <sec>
                        Timeout to shutdown kernel if connection to server is lost. Can be specified as the
                        JKS_SHUTDOWN_TIMEOUT environment variable.
  --response-server-socket-timeout <sec>
```

```
                        Socket timeout value for connecting to response server. Can be specified as the
                        JKS_RESPONSE_SERVER_SOCKET_TIMEOUT environment variable.
  --watchdog-interval <sec>
                        How often watchdog with check liveness of kernel and connection to server. Can be
                        specified as the JKS_WATCHDOG_INTERVAL environment variable.
  --kernel-script <cmd>
                        Script to run; '__connection_file__' placeholder will be substituted. Can be
                        specified as the JKS_KERNEL_SCRIPT environment variable.
  --kernel-script-base64 <cmd_base64>
                        Alternatively cmd can be in base64 format to avoid issues with templating. Can be
                        specified as the JKS_KERNEL_SCRIPT_BASE64 environment variable.
  --connection-file-dir <path>
                        Directory to store connection file json
  --encryption-key <pem_key>
                        Public key to encrypt data; must be in string in PEM format or path to .pem file.
                        Can be specified as the JKS_ENCRYPTION_KEY environment variable
```

To establish communication with the response manager, the cryptographic key stored in `JKS_ENCRYPTION_KEY` is required. This is a public key generated by response manager, used to secure communication between the response manager and `jupyter-kernel-starter`.

A typical starting command is:

```
jupyter-kernel-starter \
    --kernel-id 2a8bf66d-1577-4876-b908-a219fd4f8944 \
    --response-address 10.141.255.254:23949 \
    --shutdown-timeout 15 \
    --kernel-script 'python -m ipykernel_launcher -f __connection_file__'
```

In the preceding command `__connection_file__` is a placeholder for `jupyter-kernel-starter`. It becomes a real path to the file for `ipykernel_launcher` to start. Data from the connection file is created by jupyter-kernel-starter, and is then transferred to the response manager, so that Jupyter Kernel Manager knows how to connect to the running kernel. This way `--kernel-script` can start the kernel for any language supported by Jupyter.

Useful arguments that are often needed for more sophisticated cluster configuration are:

- `--bind-network`: used to make sure that `jupyter-kernel-starter` selects the right interface on the compute node that is connected to the login node
- `--port-range`: limits the ports that are probed to establish a connection to the response manager—this can be useful for strict network and firewall rules.

Various timeouts (`jupyter-kernel-starter --help | grep timeout`) can also be tuned.

The command options can be set in kernel templates (`templates/submit_script.sh.j2`) or directly in the created kernel.

16.5.3 Running Jupyter Kernels With Two Factor Authentication

If PAM and CMDaemon are configured with two-factor authentication (2FA), then JupyterHub needs to be instructed to support it. This can be done using `cmsh` as follows:

```
[root@basecm11 ~]# cmsh
[basecm11]% configurationoverlay
[basecm11->configurationoverlay]% use jupyterhub
[basecm11->configurationoverlay[jupyterhub]]% roles
[basecm11->configurationoverlay[jupyterhub]->roles]% use jupyterhub
```

```
[basecm11->configurationoverlay[jupyterhub]->roles[jupyterhub]]% configs
[basecm11->...]->roles[jupyterhub]->configs]% add  c.BrightAuthenticator.twofa
[basecm11->...]->roles*[jupyterhub*]->configs*[c.BrightAuthenticator.twofa*]]% set value True
[basecm11->...]->roles*[jupyterhub*]->configs*[c.BrightAuthenticator.twofa*]]% commit
```

16.5.4 Running Jupyter Kernels With Kubernetes

The Jupyter Kernel Operator (section 6.6 of the *Containerization Manual*) is the recommended way to run kernels in Kubernetes in BCM. It allows users to run unmodified images; it takes care of communication with Jupyter Kernel Provisioning as it manages the Jupyter kernel life-cycle, including cleaning up dead or orphaned kernels.

After installation and configuration by `cm-kubernetes-setup`, Jupyter Kernel Operator kernels (`k8s-cmjkop-*`) appear in the JuputerLab interface in the template section.

The following BCM templates allow users to create and run Jupyter kernels on compute nodes via Kubernetes:

Table 16.2: *BCM templates for creating and running Jupyter kernel provisioner kernels on cluster nodes via Kubernetes*

Template	Description
`k8s-cmjkop-ngc-py`	NGC images via Jupyter Kernel Operator
`k8s-cmjkop-py`	Jupyter official image via Jupyter Kernel Operator (Python)
`k8s-cmjkop-julia`	Jupyter official image via Jupyter Kernel Operator (Julia)
`k8s-cmjkop-py-spark`	Python + Spark via Jupyter Kernel Operator

The administrator has to make sure that Kubernetes is correctly configured on the cluster Kyverno is enabled and Kubernetes Permissions Manager is installed (section 4.10.2 of the *Containerization Manual*).

Details on Kubernetes installation are provided in Chapter 4 of the *Containerization Manual*.

The default configuration proposed by `cm-kubernetes-setup` is usually sufficient to run Kubernetes kernels created from BCM's templates. However, it is the responsibility of the administrator to add users registered in the Linux-PAM system to Kubernetes.

For example, a test user `jupyterhubuser` can be added to Kubernetes with:

Example

```
[root@basecm11 ~]# cm-kubernetes-setup --add-user jupyterhubuser --operators cm-jupyter-kernel-operator
```

For every new user added, `cm-kubernetes-setup` automatically generates a dedicated namespace with a name in the form <*user*>-restricted. For instance, the command in the example above, creates the namespace: `jupyterhubuser-restricted`.

Cluster administrators are strongly recommended to review the security policies for Kyverno, RBAC, and the dedicated namespaces.

In order to speed up kernel creation for the first user logging into JupyterLab when using BCM, it is recommended that all the relevant Kubernetes images are pre-loaded on the compute nodes that Jupyter Kernel Provisioning can contact.

16.5.5 Running Jupyter Kernels Based On NGC Containers

Jupyter NGC templates are available in the list of templates if:

- Kubernetes is set up
- NVIDIA GPUs are available on the Kubernetes cluster
- Jupyter Kernel Operator (section 6.6 of the *Containerization Manual*) is installed

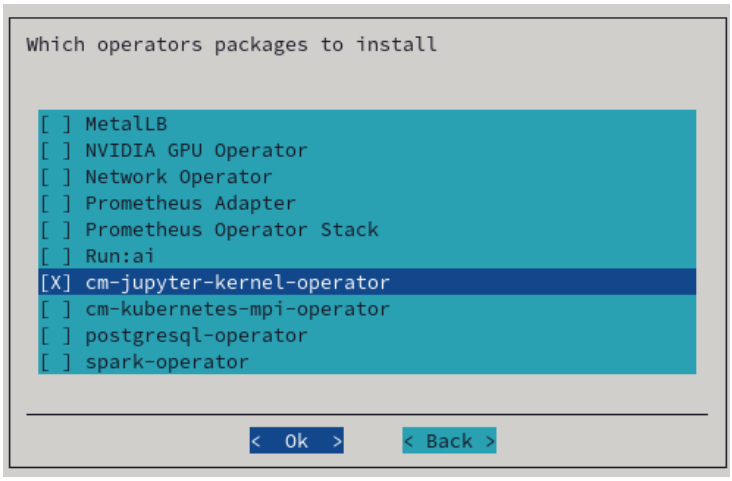


Figure 16.8: Jupyter Kernel Operator selection with `cm-kubernetes-setup`

It is also strongly advised to enable Kyverno on the Kubernetes cluster.

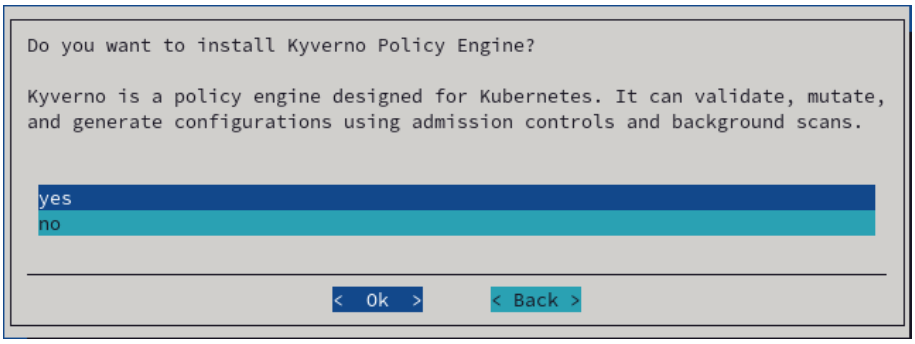


Figure 16.9: Kyverno policy engine selection in `cm-kubernetes-setup`

The user needs to be given permission to access to the Jupyter Kernel Operator. This can be done via the `cm-kubernetes-setup` TUI, by adding a user, and then setting the permissions in the permissions screen that shows up (figure 16.10):

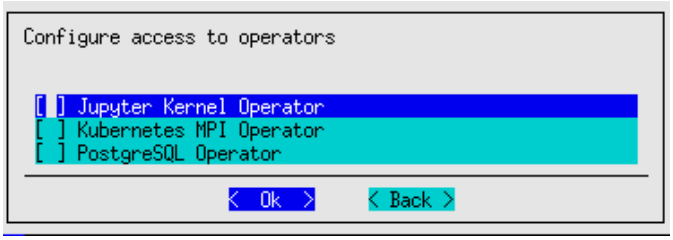


Figure 16.10: User permissions in `cm-kubernetes-setup`

The permissions can also be configured using the CLI:

```
# cm-kubernetes-setup --add-user=alice --operators=cm-jupyter-kernel-operator
```

Once Jupyter kernel Operator is available, then kernel templates appear in the list:



Figure 16.11: Jupyter Kernel Operator templates

A kernel can be created from the template:

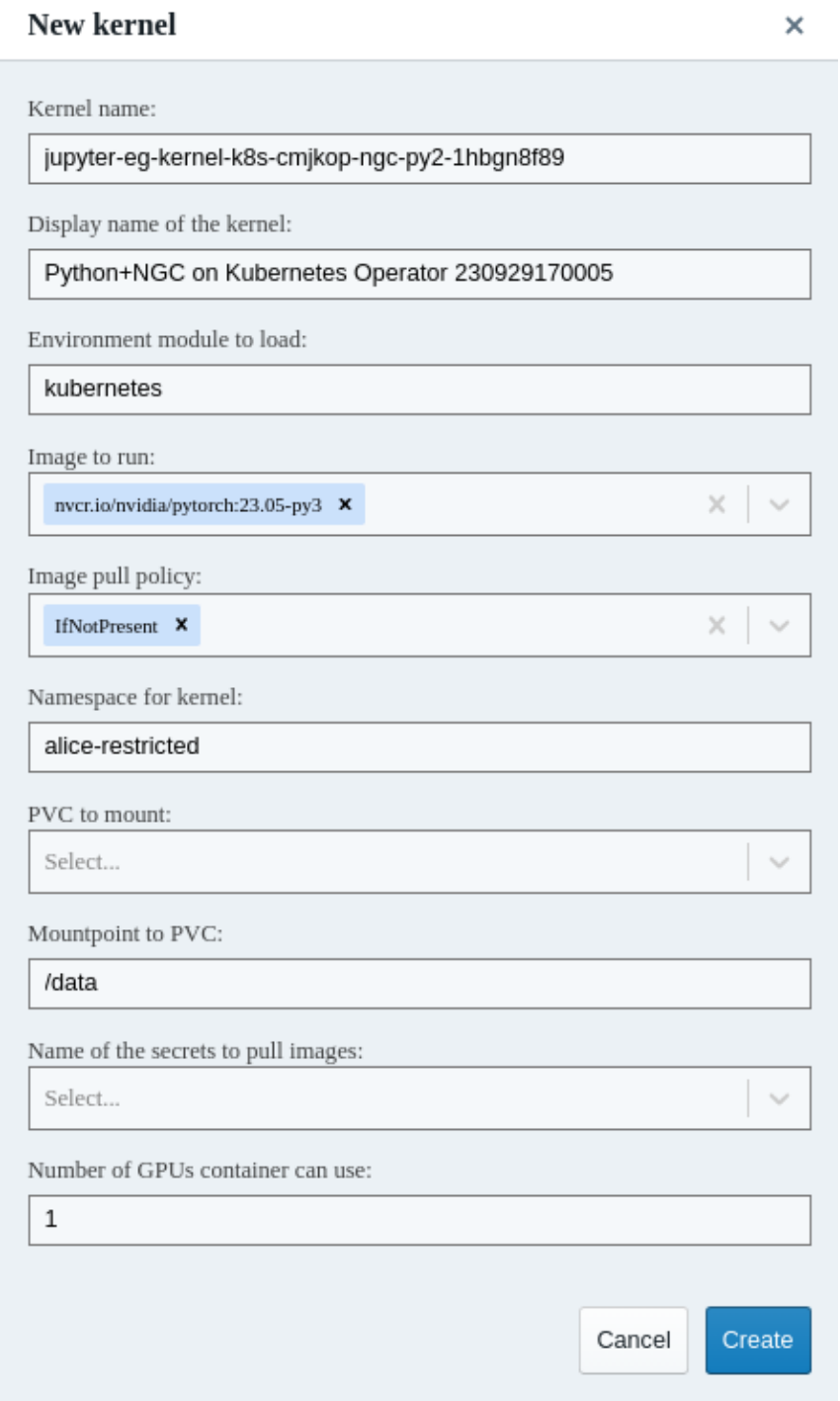


Figure 16.12: Creating kernel from template

If authentication is required to access container images from the private registry, then the template can be modified as described in section 6.6.8 of the *Containerization Manual*.

16.5.6 Running Jupyter Kernels With Workload Managers

BCM's Jupyter Kernel Provisioning kernels can be created and run by users on compute nodes via workload managers (WLMs). For convenience, Slurm is used as an example in this section. However, the same instructions are valid for the other WLMs listed in table 16.1.

The templates used to create Slurm kernels are BCM's

- slurm-bash and
- slurm-py312,

which offer a Bash and a Python 3.12 environment respectively.

The administrator has to make sure that Slurm is correctly configured on the cluster. Slurm installation is described in section 7.3.

The default configuration that cm-wlm-setup suggests in Express mode is usually sufficient to run Slurm kernels created from BCM's templates. If Slurm is configured to automatically detect GPUs, they will be listed as available resources while instantiating Slurm templates.

The Jupyter login node must be authorized to submit Slurm jobs. This is typically the case, since the JupyterHub login node is by default the head node, and cm-wlm-setup by default assigns the slurmsubmit role to the head node.

Finally, the administrator must make sure that relevant dependency packages are installed on the software image used by Slurm clients. The image used is typically for the compute nodes. Missing packages may cause kernels to fail at startup or at run time.

In particular, administrators need to install cm-python312 to use kernels based on the slurm-py312 template.

16.6 Jupyter Kernel Creator Extension Customization

The Jupyter kernel templates described in section 16.5 are stored under the directory /cm/shared/apps/jupyter/current/share/jupyter/kerneltemplates/.

The template of a particular Jupyter kernel in BCM's Jupyter Kernel Creator extension is a directory containing at least two files: meta.yaml and kernel.json.j2. The kernel template can also contain other files placed in the same directory, such as icons. These files are copied to the target user kernels directory upon kernel creation, after the template is instantiated.

The meta.yaml file includes all the parameters that can be substituted into kernel.json.j2 and is defined with the YAML format (https://yaml.org/).

The kernel.json.j2 file is a skeleton of the kernel.json file to be generated. It can contain some placeholders, and is defined with the Jinja2 format (https://jinja.palletsprojects.com/).

16.6.1 Kernel Template Parameters Definition

The kernel template meta.yaml file defines all the parameters that can be used in kernel.json.j2.

It should contain three entries:

- display_name: the name that will be displayed for the kernel in the JupyterLab Launcher;
- features: a list of features that must be available on the cluster to show this kernel template to JupyterLab users in the BCM extensions panel;
- parameters: the variables for kernel.json.j2.

The display_name entry is an arbitrary string:

```
[...]
display_name: "A simple kernel"
[...]
```

The features entry is a string that defines the condition under which the template becomes available in the user interface. The features definitions are found within filter.yaml, which is in the same directory as the kernel templates (section 16.5.1)

For example, a kernel template that only requires Kubernetes to work contains this line in its meta.yaml file:

```
[...]
features: features: "k8s-jupyter-operator-enabled and k8s-nvidia-gpu-available"
[...]
```

and `filter.yaml` must have definitions of both conditions:

```
features:
[...]
  k8s-jupyter-operator-enabled:
    getter: shell
    timeout: 5
    exec:
      - "source /etc/profile.d/modules.sh"
      - "module load kubernetes"
      - "kubectl get cmjupyterkernels"
[...]
  k8s-nvidia-gpu-available:
    getter: shell
    timeout: 5
    exec:
      - "source /etc/profile.d/modules.sh"
      - "module load kubernetes"
      - "kubectl get nodes -o \\"
      - "jsonpath=\"{.items[*].status.capacity['nvidia\\.com/gpu']}\" \\"
      - "| egrep -q '.'"
[...]
```

The preceding means that a user can list `cmjupyterkernels` objects in Kubernetes, and that nodes must have `nvidia.com/gpu` records available in their manifest

The scripts (the `exec` entries) in the preceding filters use typical values. For example, default values for the Kubernetes environment module. These lines can be edited to be consistent with the actual situation on the cluster.

The same principle and customizations should be considered for the other workload managers.

Finally, the `parameters` entry contains a dictionary of kernel parameters. The keys of the dictionary are parameter names that are used in `kernel.json.j2`. The values of the dictionary are options that help users choose a correct value for the parameter. Parameter options are also dictionaries.

A kernel template `meta.yaml` is thus structured as:

```
[...]
parameters:
  <parameter_name1>:
    <option_key>: <option_value>
    ...
    <option_key>: <option_value>
  <parameter_name2>:
    <option_key>: <option_value>
    ...
    <option_key>: <option_value>
  ...
  <parameter_nameN>:
    <option_key>: <option_value>
    ...
    <option_key>: <option_value>
[...]
```

Parameter names are arbitrary strings.

Option keys are strings. The only possible values for these strings are:

- `type`
- `definition`
- `limits`

The option keys `type` and `definition` are mandatory. The option key `limits` is optional.
For example, a kernel template with two parameters `foo` and `bar` looks as follows:

Example

```
[...]
parameters:
  foo:
    type: <option_value>
    definition: <option_value>
    limits: <option_value>
  bar:
    type: <option_value>
    definition: <option_value>
    limits: <option_value>
[...]
```

The `type` Option

The `type` option key defines the type of the kernel parameter.

The `type` option key only accepts one of the following string values:

- `num`: for numeric values (both float and integer are supported);
- `str`: for arbitrary strings;
- `bool`: for boolean values (a checkbox is presented in the user interface);
- `oneof`: for picker to choose one value out out of several provided;
- `list`: for lists of pre-defined or dynamically-generated settings;
- `uri`: for interactive RESTful endpoints (only `/kernelcreator/envmodules` is currently supported).

The `definition` Option

The `definition` option key defines how the parameter value is retrieved and displayed.

The `definition` option key accepts dictionary-like values. Allowed string keys for the dictionary are:

- `display_name`: the name that is displayed for the parameter in the kernel customization dialog. It accepts an arbitrary string as a value;
- `getter`: how parameter values are retrieved. It only accepts as value one of the following strings:
  - `static`: the values for the parameter are pre-defined;
  - `shell`: the values for the parameter are the output of the shell script;
  - `python`: the values for the parameter are the output of a Python script;
- `values`: possible values for the parameter that are displayed in the kernel customization dialog. It accepts a list of arbitrary values;
- `default`: default value for the parameter. It accepts a value from `values`;
- `exec`: the script to be executed to fill `values` when `getter` is `shell` or `python`. It accepts a shell/Python script.

If `type` option value is `list`, then every line of `getter` with a setting of `shell` or `python` is treated as an element of the list.

The `limits` Option

The `limits` option key offers a way to apply bounds on values provided by users. This usually reduces the chances of making mistakes and helps users defining correct kernels before actually running them.

The `limits` option key accepts dictionary-like values according to the value chosen for the `type` option key.

If `num` is the `type`, then `limits` can contain:

- `min`: the minimum numeric value;
- `max`: the maximum numeric value.

If `str` or `list` is the `type`, then `limits` can contain:

- `min_len`: the minimum length of the string or the list;
- `max_len`: the maximum length of the string or the list.

For example, if every node in the cluster has no more than 4 GPUs, then the upper limit on the requested GPU number can be set to 4:

```
[...]
parameters:
  gpus:
    type: num
    definition: <option_value>
    limits:
      min: 1
      max: 4
[...]
```

These limits are not a security measure. They should be considered as convenient sanity checks for values entered while instantiating a kernel template. This is because users are always able to later directly edit the generated `kernel.json`, thereby ignoring such limits.

16.6.2 Kernel Template Parameters Usage

During the creation of the kernel definition from the template, the `kernel.json` file is created from `kernel.json.j2`, so all the Jinja2 variables are substituted based on the `meta.yaml` file and choices provided by the user. When a kernel startup is initiated, Jupyter Kernel Provisioning uses `kernel.json` and some internal variables to create WLM jobs or Kubernetes manifests from templates defined in the `kernel.json` file.

Variables are available from a `vars` dictionary on kernel startup. These variables are defined within a subsection of `metadata.kernel_provisionersection`. The subsection is `config.<`*subsection*`>.vars`, where <*subsection*> depends on the chosen kernel provisioner.

Other dictionaries are `internal` and `env`.

The `internal` dictionary contains several variables defined at runtime:

- `uuid`: Unique ID for every running kernel process. Does not persist across kernel restarts.
- `kernel_id`: ID of the kernel. Persists across restarts.
- `kernel_spec_resource_dir`: Full path to the kernel spec definition, normally: `~/.local/share/jupyter/kernel-name`.
- `username`: PAM username of the user starting the kernel.
- `uid`: PAM user's UID.
- `gid`: PAM user's GID.

- `homedir`: User's home directory from the PAM database.
- `shell`: User's shell from the PAM database.
- `kernel_cmd`: Command to run. For most kernels it is set to an empty string, as the actual command is defined separately in WLM job or Kubernetes manifest templates.

The `env` variables pass some environment variables matching a regex in order to pass some variables to be used in accessing API endpoints. Users thus do not need to hardcode API keys for notebooks.

16.6.3 Filtering Out Irrelevant Templates From The Interface For Users

The list of kernel templates that are available for users of the Jupyter Kernel Creator can be modified as follows:

In the parent directory of the templates, the file `filter.yaml` describes items called `features`. These can be statically defined or they can represent scripts to be executed. The result of an executed script can be `true` or `false`. If the script code exits with zero, then the result is `true`.

Every kernel template can have the `feature` field, which is a boolean expression that is calculated when the JupyterLab extension is started.

Example

```
# cat filter.yaml
...
  slurm:
    getter: shell
    timeout: 5
    exec:
      - "source /etc/profile.d/modules.sh"
      - "module load slurm"
      - "sinfo"
...

# cat slurm-py312/meta.yaml
...
features: "slurm"
...
```

In the preceding example, the `slurm-py312` template is shown in the JupyterLab interface if the script in the `exec:` section is able to finish successfully.

It is also possible to define more complicated rules:

Example

```
# cat filter.yaml
features:
  kubernetes:
    getter: shell
    timeout: 5
    exec:
      - "source /etc/profile.d/modules.sh"
      - "module load kubernetes"
      - "kubectl get pods"
  k8s-jupyter-operator-installed:
    getter: shell
    timeout: 5
    exec:
```

```
      - "source /etc/profile.d/modules.sh"
      - "module load kubernetes"
      - "(kubectl get cmjupyterkernels 2>&1 || true) \\"
      - "| egrep -q '^Error from server \\(Forbidden\\)'"
  k8s-jupyter-operator-enabled:
    getter: shell
    timeout: 5
    exec:
      - "source /etc/profile.d/modules.sh"
      - "module load kubernetes"
      - "kubectl get cmjupyterkernels"
...

# cat jupyter-eg-kernel-k8s-py/meta.yaml
---
display_name: "Python on Kubernetes"
features: "kubernetes and not k8s-jupyter-operator-installed and not k8s-jupyter-operator-enabled"
...
```

In the preceding example, `jupyter-eg-kernel-k8s-py` is shown when Kubernetes is installed, and is hidden if Jupyter Kubernetes Operator is available for the user.

The `filter.yaml` file also supports statically defined features:

Example

```
features:
  always-enabled:
    getter: static
    default: True
  always-disabled:
    getter: static
    default: False
...
```

and supports Python code:

Example

```
  python3-available:
    getter: python
    timeout: 5
    exec:
      - "import sys"
      - "sys.exit(0) if sys.version_info.major == 3 else sys.exit(1)"
...
```

The Kubernetes Jupyter Kernel Operator is discussed in section 6.6 of the *Containerization Manual*.

16.7 Jupyter VNC Extension

16.7.1 What Is Jupyter VNC Extension About?

VNC (Virtual Network Computing) is a screen sharing service that can work in a browser.

If VNC is allowed by the cluster administrator, then the Jupyter environment configured by BCM can be used to start and control remote desktops via VNC with the *Jupyter VNC* extension.

Several kernels created from BCM's templates are capable of running VNC sessions so that users can run GUI applications. In order to do so, the cluster nodes where kernels are executed must support VNC.

During the `cm-jupyter-setup` run, the TUI prompts for installation of VNC servers on the nodes. The cluster administrator can select the nodes on which it installs. On these nodes Jupyter kernels can work with several types of VNC clients.

16.7.2 Enabling User Lingering

User lingering is a systemd setting that sets a user manager for a user at boot and keeps it around after logout. This allows that user to run long-running sessions despite not being logged in. Enabling user lingering may be required for Jupyter VNC extension to run for a relatively complicated desktop such as KDE or GNOME.

For each user on each machine where these environments are installed, the following command must be run:

```
loginctl enable-linger <username>
```

The command may also be carried out using prolog/epilog scripts in the chosen WLM.

16.7.3 Starting A VNC Session With The Jupyter VNC Extension

Users can start a VNC session with the button added by Jupyter VNC (figure 16.13). Additional VNC parameters can be optionally specified.

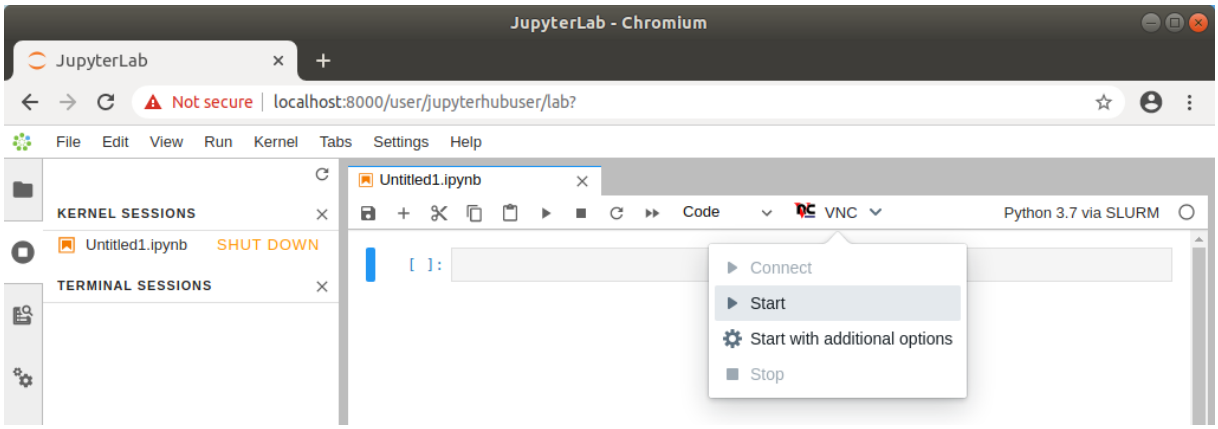


Figure 16.13: Starting Jupyter VNC session from kernel

If VNC is available and correctly configured on the node where the kernel is running, then a new tab is automatically created by Jupyter VNC containing the new session (figure 16.14). A user can now freely interact in JupyterLab both with the notebook and with the desktop environment.

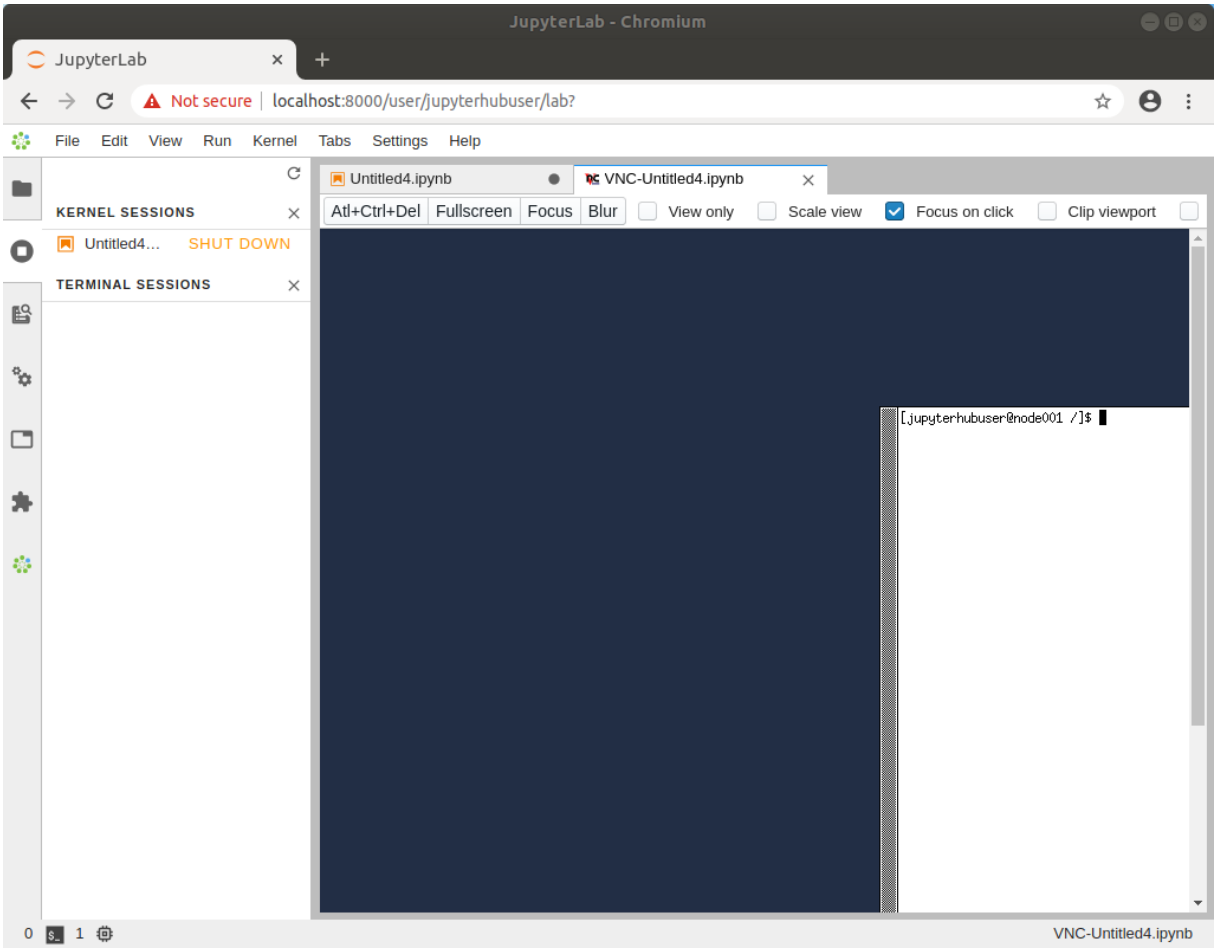


Figure 16.14: Running Jupyter VNC session from kernel

To provide a user-friendly experience, Jupyter VNC also allows the graphical viewport to be resized, so that the desktop application can run full-screen (figure 16.15).

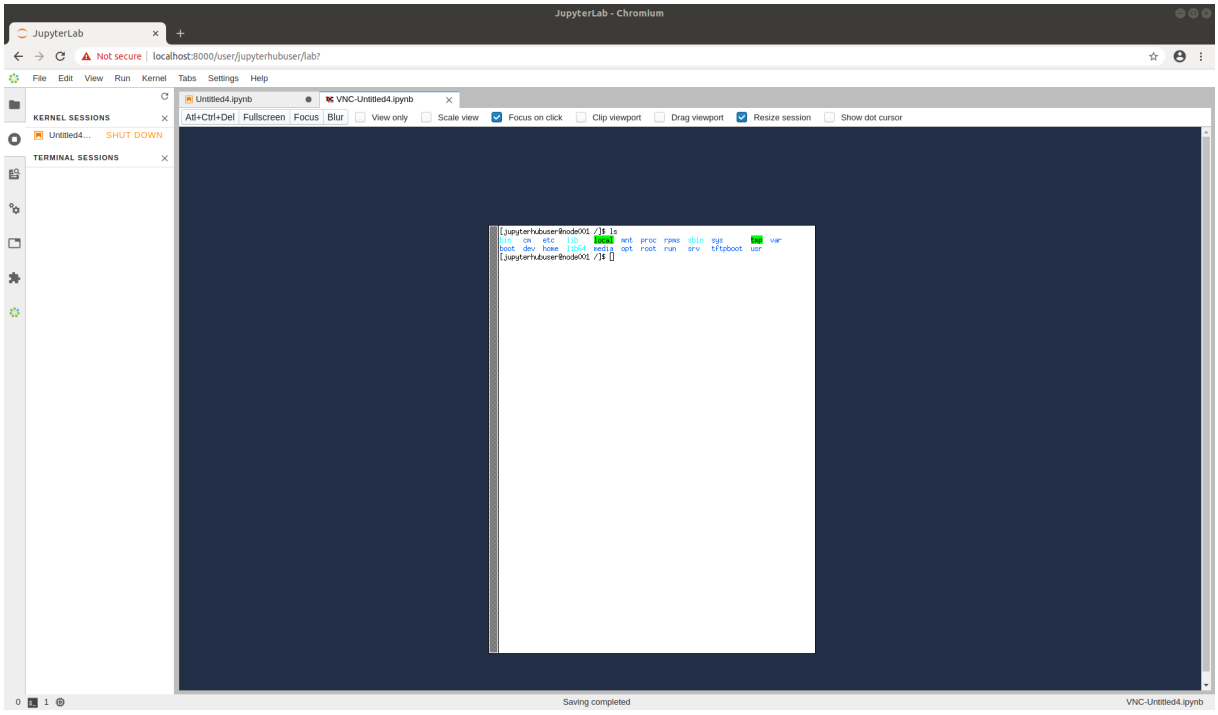


Figure 16.15: Running Jupyter VNC session from kernel (full-screen)

16.7.4 Running Examples And Applications In The VNC Session With The Jupyter VNC Extension

Once the VNC session is correctly started and the new JupyterLab tab has been created, Jupyter VNC automatically exports the `DISPLAY` environment variable to the running notebook (figure 16.16). Doing so means that any application or library running in the notebook can make use of the freshly created desktop environment. An example of such a library is OpenAI Gym, a toolkit for developing and comparing reinforcement learning algorithms, that is distributed by BCM.

Among the examples distributed by BCM (section 16.3), a notebook running PyTorch in the OpenAI Gym CartPole environment can be found. If executed after a VNC session has been started, a user can then observe the model being trained in real time in the graphical environment.

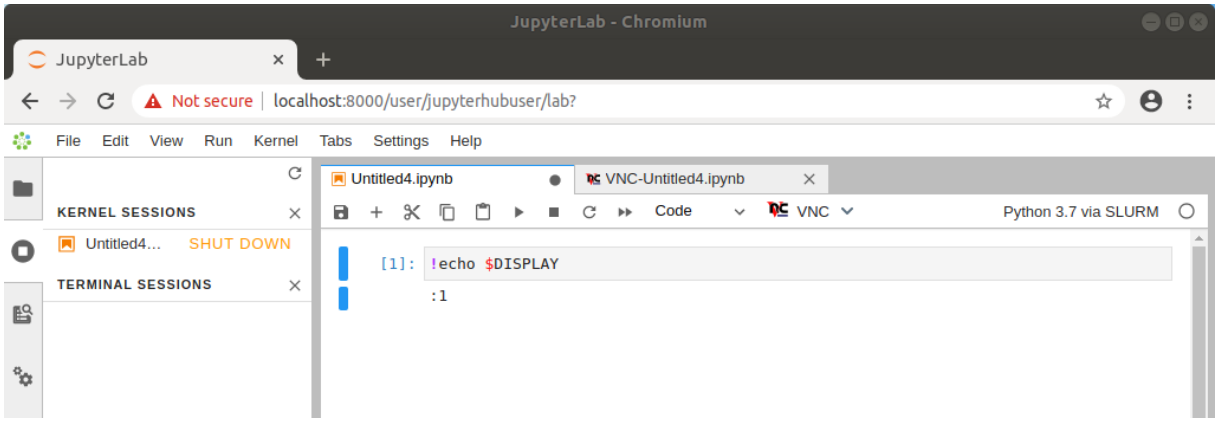


Figure 16.16: Automatic configuration of DISPLAY environment variable

16.8 Jupyter WLM Magic Extension

In the Jupyter environment configured by BCM, the *Jupyter WLM Magic* extension can be used to schedule workload manager jobs from notebooks.

The Jupyter WLM Magic extension is an IPython extension. It is designed to improve the capabilities of Jupyter's default Python 3 kernel, which runs on the login node.

The Jupyter WLM Magic extension should therefore not be used from kernels running on compute nodes, such as those typically created with BCM's Jupyter Kernel Creator extension (section 16.5), and submitted via Jupyter Kernel Provisioning. Indeed, compute nodes running these kernels are often incapable of starting workload manager jobs in many default WLM configurations.

Jupyter WLM Magic extension makes it possible for users to programmatically submit WLM jobs, and then interact with their results. This can be done while using the Python programming language and its libraries, which are available in the notebook.

Users submit jobs and check their progress from the login node. The actual computation is distributed by the underlying workload manager across compute nodes, which means that server resources are spared.

Jupyter WLM Magic commands are available in the IPython kernel as *magic functions* (`https://ipython.readthedocs.io/en/stable/interactive/tutorial.html#magic-functions`). A new line magic (`%`) and a new cell magic (`%%`) are now added in the kernel, according to the workload manager:

- Platform LSF: `%lsf_job` and `%%lsf_job`
- Slurm: `%slurm_job` and `%%slurm_job`

A user can list the magic functions in the kernel to see if they are available, with Jupyter's builtin command `%lsmagic` (`https://ipython.readthedocs.io/en/stable/interactive/magics.html#magic-lsmagic`):

Example

```
In  []: %lsmagic
Out []: root:
            line:
                automagic:"AutoMagics"
                autocall:"AutoMagics"
                [...]
                slurm_job:"SLURMMagic"
                lsf_job:"LSFMagic"
            cell:
                js:"DisplayMagics"
                javascript:"DisplayMagics"
                [...]
                slurm_job:"SLURMMagic"
                lsf_job:"LSFMagic"
```

The magic functions introduced by this BCM extension share a similar syntax. For convenience, Slurm is used as an example in this section. However, the same instructions are valid for the other WLMs.

Users can check which options are available for a WLM function with the line magic helper:

Example

```
In  []: %slurm_job --help
Out []: usage: %slurm_job [-h] [--module MODULE] [--module-load-cmd MODULE_LOAD_CMD]
                          [--shell SHELL] [--submit-command SUBMIT_COMMAND]
                          [--cancel-command CANCEL_COMMAND]
```

```
                        [--control-command CONTROL_COMMAND]
                        [--stdout-file STDOUT_FILE] [--stderr-file STDERR_FILE]
                        [--preamble PREAMBLE] [--timeout TIMEOUT]
                        [--check-condition-var CHECK_CONDITION_VAR]
                        [--job-id-var JOB_ID_VAR]
                        [--stdout-file-var STDOUT_FILE_VAR]
                        [--stderr-file-var STDERR_FILE_VAR] [--dont-wait]
                        [--write-updates WRITE_UPDATES]
                        [--check-status-every CHECK_STATUS_EVERY]

      optional arguments:
        -h, --help            show this help message and exit
      [...]
```

Line magic functions are typically used to set options with a global scope in the notebook. By doing so, a user will not need to specify the same option every time a job will be submitted via cell magic. For example, if two Slurm instances are deployed on the cluster and their associated environment modules are `slurm-primary` and `slurm-secondary`, a user could run the following line magic once to configure the Jupyter WLM Magic extension to always use the second deployment:

Example

```
In  []: %slurm_job --module slurm-secondary
Out []:
```

Now, jobs will always be submitted to `slurm-secondary`. This is more convenient than repeatedly defining the same module option for every cell magic upon scheduling a job:

Example

```
In  []: %%slurm_job --module slurm-secondary
        <WLM JOB DEFINITION>
Out []: <WLM JOB OUTPUT>
In  []: %%slurm_job --module slurm-secondary
        <WLM JOB DEFINITION>
Out []: <WLM JOB OUTPUT>
```

It should be noted that line magic functions cannot be used to submit WLM jobs. Cell magic functions have to be used instead.

A well-defined cell contains the WLM cell magic function provided by the extension, followed by the traditional job definition. For example, a simple MPI job running on two nodes can be submitted to Slurm by defining and running this cell:

Example

```
In  []: %%slurm_job
        #SBATCH -J mpi-job-example
        #SBATCH -N 2
        module load openmpi
        mpirun hostname
Out []: COMPLETED
        STDOUT file content: /home/demo/.jupyter/wlm_magic/slurm-1.out
        node001
        node001
        node002
        node002
```

Users can take advantage of the Jupyter WLM Magic extension to store some information into Python variables about the job being submitted. The information could be the ID or the output file name, for example. Users can then later programmatically interact with them in Python. This feature is convenient when a user wants to, for example, programmatically carry out new actions depending on the job output:

Example

```
In  []: %%slurm_job --job-id-var my_job_id --stdout-file-var my_job_out
        #SBATCH -J mpi-job-example
        #SBATCH -N 2
        module load openmpi
        mpirun hostname
Out []: COMPLETED
        STDOUT file content: /home/demo/.jupyter/wlm_magic/slurm-2.out
        node001
        node001
        node002
        node002

In  []: print(f"Job id {my_job_id} was written to {my_job_out}")
        print(f"Output lines: {open(my_job_out).readlines()}")
Out []: Job id 2 was written to /home/demo/.jupyter/wlm_magic/slurm-2.out
        Output lines: ['node001\n', 'node001\n', 'node002\n', 'node002\n']
```

Users can also exploit Python variables to define the behavior of the Jupyter WLM Magic extension. For example, they can define a Python boolean variable to submit a WLM job only if a condition is true:

Example

```
In  []: run_job = 1 == 2
Out []:

In  []: %%slurm_job --check-condition-var run_job
        #SBATCH -J mpi-job-example
        #SBATCH -N 2
        module load openmpi
        mpirun hostname
Out []: Variable run_job is 'False'. Skipping submit.
```

16.9 Jupyter Kubernetes Operators Manager

The Jupyter Kubernetes Operators Manager tool is designed to make it easier to handle everyday tasks involving Kubernetes.

Some of the items that the tool can manage are:

- Pods
- PostgreSQL databases
- Spark tasks (jobs that process a lot of data)
- Persistent Volume Claims (requests for specific storage space). Particularly handy is the ability to move data between user folders and Persistent Volumes (special types of storage space in Kubernetes).

Jupyter Kubernetes Operators Manager can be accessed as follows in the Jupyter Notebooks web interface (figure 16.17):

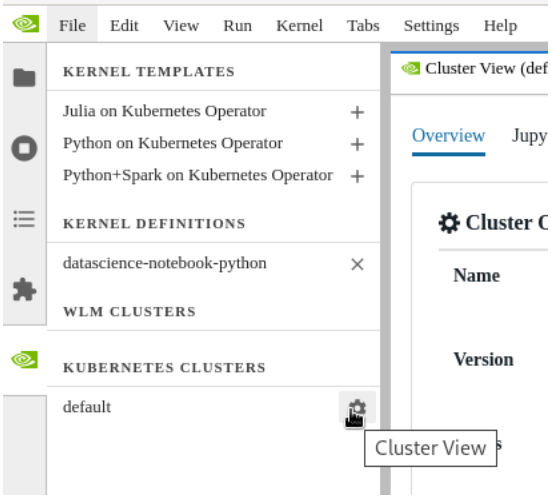


Figure 16.17: Jupyter Kubernetes Operators Manager: Kubernetes cluster list and selection

1. The NVIDIA Base Command Manager logo (on the left side of the screen,) is clicked. A list of Kubernetes clusters is then displayed.
2. The `Cluster View` button is used to select the cluster that is to be worked with.

Jupyter Kubernetes Operators Manager displays resources and objects via a tabbed view. This view is for the restricted namespace to which the user has access. With Kubernetes as set up by BCM, this is the namespace of the form `<user>-restricted`.

16.9.1 Overview Tab

The `Overview` tab (figure 16.18) provides a high-level overview of the current state of the selected Kubernetes cluster.

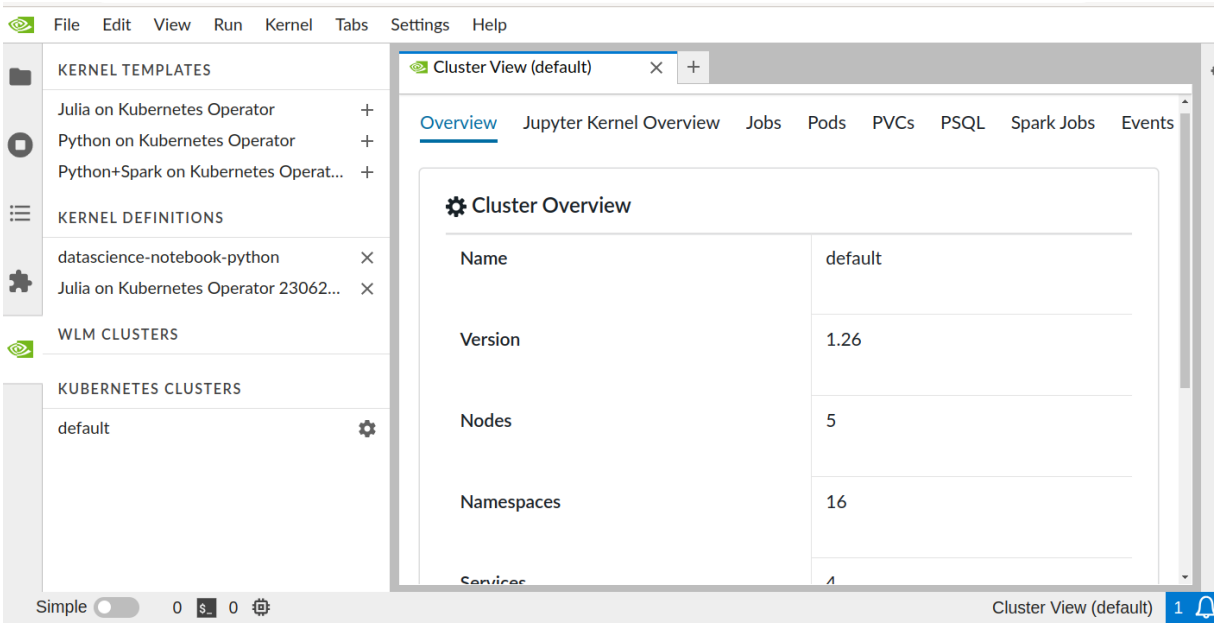


Figure 16.18: Jupyter Kubernetes Operators Manager: overview

Among other useful details, it displays:

- the cluster name
- the Kubernetes version that is running
- the number of namespaces and pods

16.9.2 Jupyter Kernel Overview Tab

The `Jupyter Kernel Overview` tab (figure 16.19) lists active Jupyter kernel instances along with their associated events.

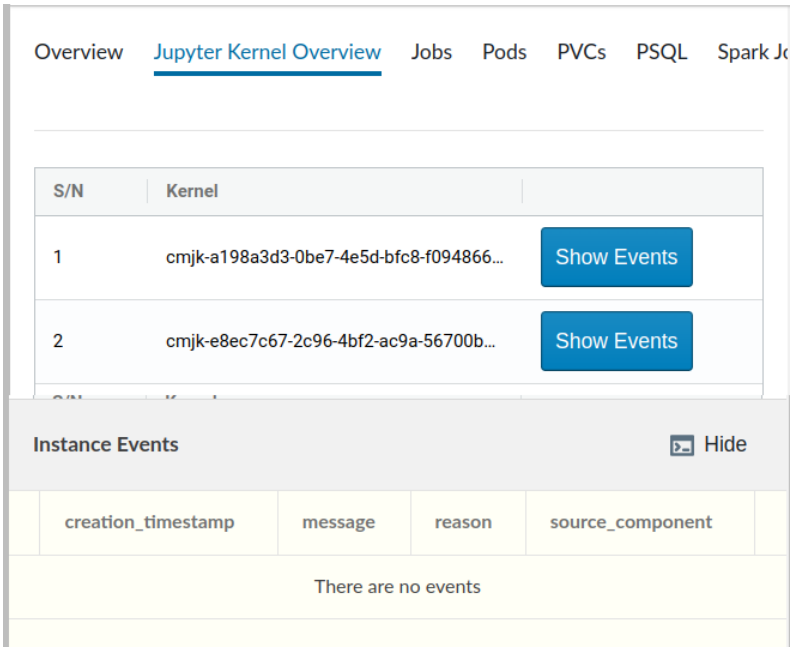


Figure 16.19: Jupyter Kubernetes Operators Manager: Jupyter kernel overview tab

After a kernel is stopped or removed, these events can be found under the `Events` tab. Events are discussed later on.

16.9.3 Jobs Tab

The `Jobs` tab (figure 16.20) allows a data migration job to be run. A data migration job manages the transfer of data between the user-accessible directories of the filesystem and the Persistent Volumes. The transfer can be managed in either direction.

The data transfer is needed to enable access to the data for Spark instances, and for situations where pods need to be run under a different user ID/group ID (UID/GID) from the original user.

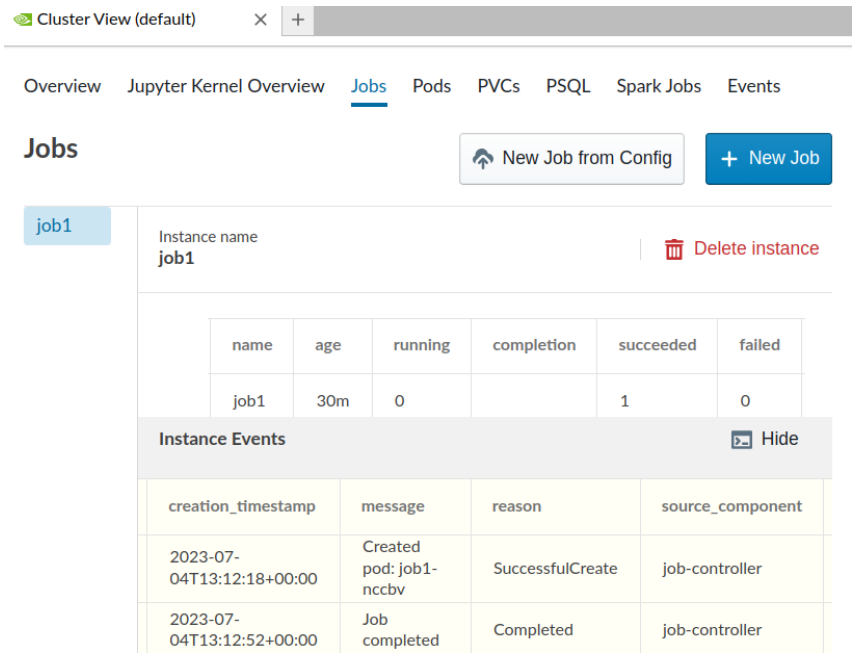


Figure 16.20: Jupyter Kubernetes Operators Manager: Jupyter data migration jobs tab

To initiate a data migration job, several fields must be set. This can be done via a pop-up dialog, that comes up on clicking either of these buttons:

- `New job from Config`: allows a YAML file configuration to be submitted
- `New Job`: allows direct configuration (figure 16.21)

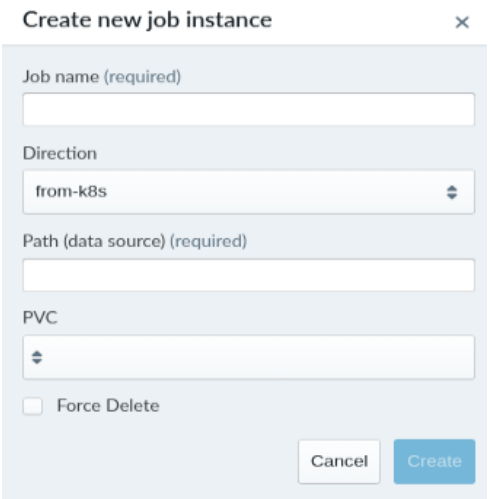


Figure 16.21: Jupyter Kubernetes Operators Manager: data migration jobs creation

Fields that may be set include:

- **Job name**: specifies the name of the migration job.
- **Direction**: sets the direction of the data migration.
  - `to-k8s`: copies data to the Persistent Volume Claim (PVC) from the file system
  - `from-k8s`: copies data from the PVC back to the file system

- **Path**: provides the path on the filesystem to/from where the data is migrated
- **PVC**: sets the name of the Persistent Volume Claim that is involved in the data migration.

Optionally, the `Force Delete` checkbox can be ticked. If ticked, then the migration job deletes data from the target location if the corresponding file or directory does not exist in the source location.

In particular, if the source location is completely empty, then setting `Force Delete` results in the removal of all target data. Setting `Force Delete` should therefore be done with caution due to the significant data loss that it can cause.

The execution status of a job can be monitored in the status and events area. This section displays the number of worker pods and their respective statuses, which include:

- **Running**: the number of pods currently in operation.
- **Completion**: the desired number of pods that should successfully complete the job. Not applicable for migration jobs, but it can be used if a custom manifest is specified, as described later.
- **Succeeded**: The number of pods that have successfully completed their tasks.
- **Failed**: The number of pods that have failed to complete their tasks.

Additionally, a custom job manifest can be uploaded from the user's workstation if required.

16.9.4 Pods Tab

The `Pods` tab (figure 16.22) displays user pods, reads stdout, and shows associated events from pods.

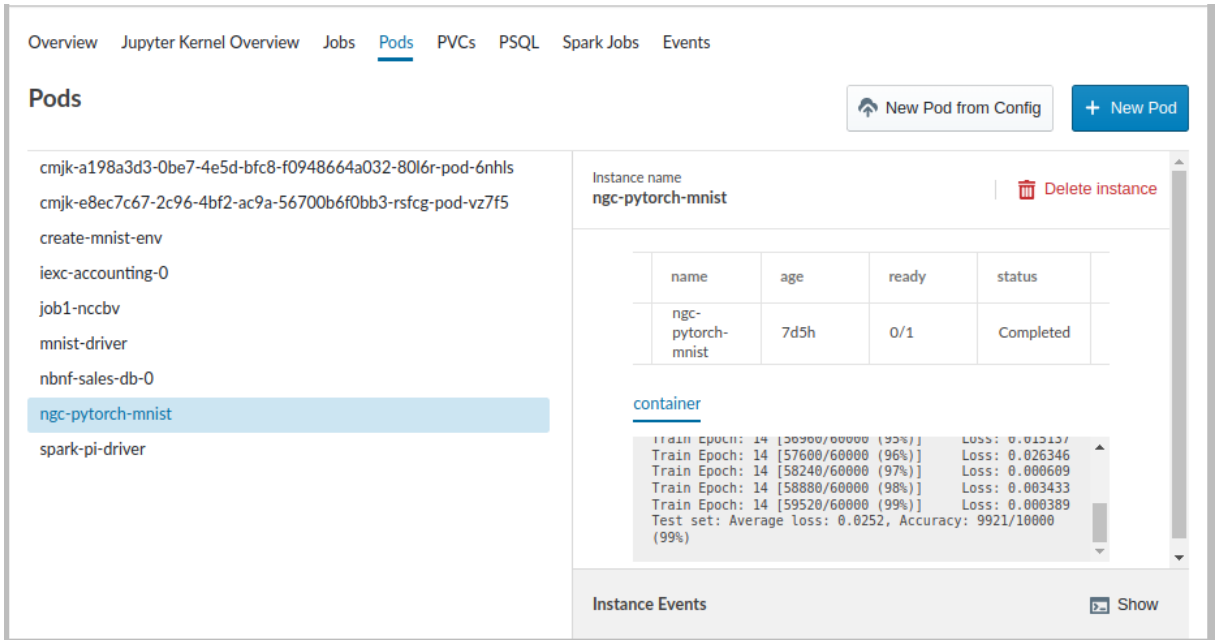


Figure 16.22: Jupyter Kubernetes Operators Manager: pods overview tab

The tab also allows pod creation from a custom definition manifest uploaded from the user's workstation, or by filling out a simplified form with commonly used fields:

1. **Name**: name of the Pod.
2. **Command**: program and its arguments that are to run inside the Pod.
3. **Image**: container image.

4. **Host Path** (Optional): path to mount from the filesystem. If used, the UID/GID of the running process matches the UID/GID of the user.
5. **Environment Variables** (Optional): These variables are set for the process inside the running Pod.
6. **PVCs** (Optional): A list of Persistent Volume Claims and the path to mount them inside the Pod.

In figure 16.23 `main.py` has been downloaded earlier from the official PyTorch repository and placed in the user's home directory using `wget`:

```
wget https://raw.githubusercontent.com/pytorch/examples/main/mnist/main.py
```

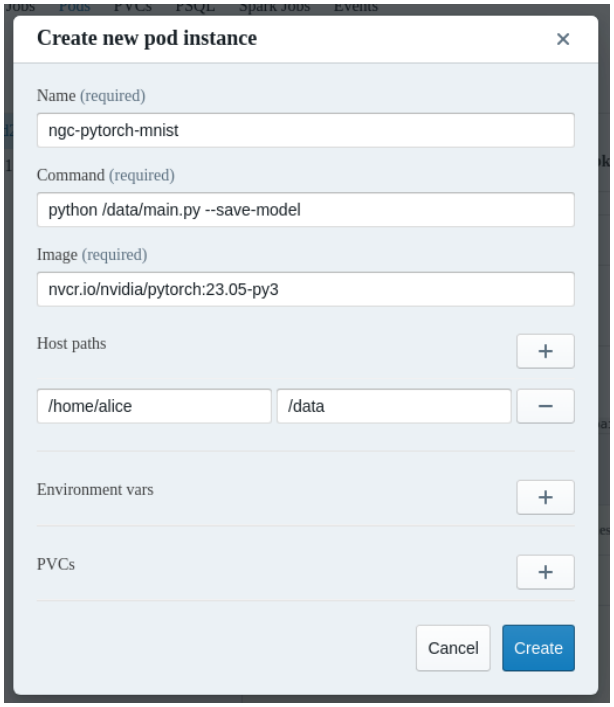


Figure 16.23: Jupyter Kubernetes Operators Manager: pod creation

16.9.5 PVCs Tab

A Persistent Volume Claim (PVC) is a user's request for a specific amount of storage space within a Kubernetes cluster with defined characteristics.

The `PVCs` tab (figure 16.24) displays an overview of the PVCs.

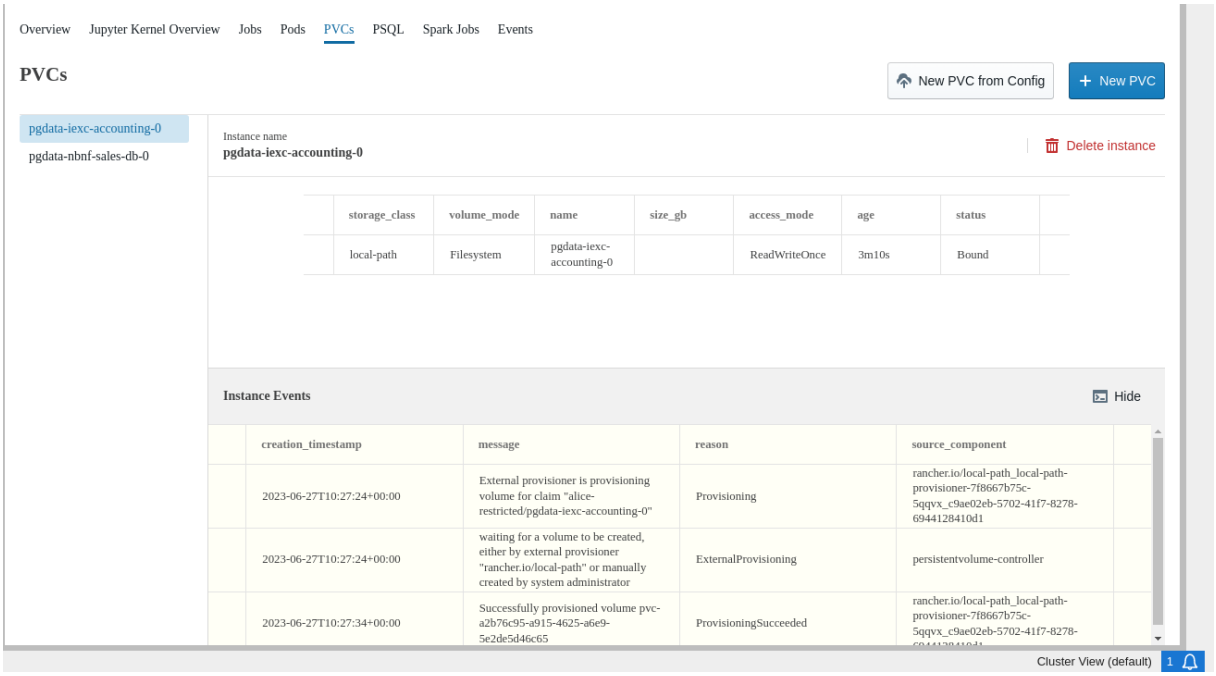


Figure 16.24: Jupyter Kubernetes Operators Manager: PVCs overview tab

The tab allows users to manage existing PVCs or create new ones via a dialog.

To create a new PVC, several fields must be set. This can be done via a pop-up dialog, that comes up on clicking either of these buttons:

- the `New PVC from Config` button: This allows a YAML file configuration to be submitted to set the fields.
- the `New PVC` button: This allows the configuration to be set directly (figure 16.25).

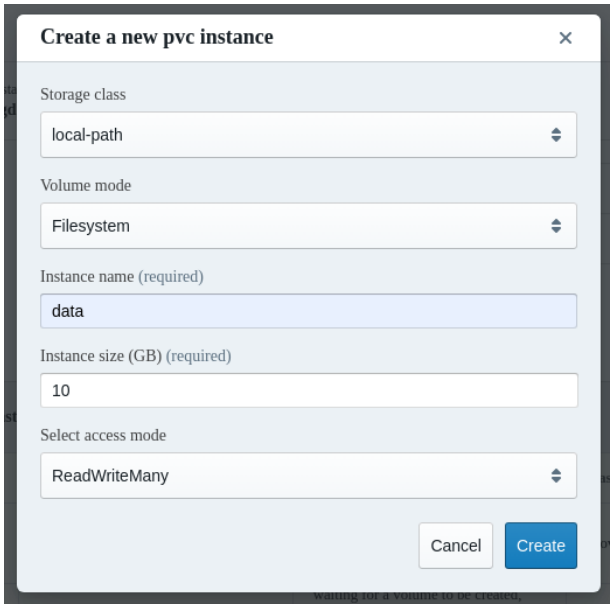


Figure 16.25: Jupyter Kubernetes Operators Manager: PVC creation

To create a new PVC directly, the following fields need to be set in the form:

1. **Storage class**: The storage class is `local-path` for now in BCM version 11.
2. **Volume mode**: can be either
   - `Filesystem`, where the requested space is accessible as a formatted filesystem, or
   - `Block`, where the space is accessible as a raw block device.

   For the `local-path` storage class, only `Filesystem` is available.
3. **Instance name**: arbitrary name of the PVC.
4. **Instance size (GB)**: requested size. It can be ignored for `local-path`, as quotas are not supported for shared filesystems. The resulting volumes for a cluster <*cluster_name*> are located under `/cm/shared/apps/kubernetes/`<*cluster_name*>`/var/volumes` and are configurable during cluster setup.
5. **Select access mode**:
   - **ReadWriteOnce**: The volume can be mounted as read-write by a single node. Multiple pods can still access the volume if they are running on the same node.
   - **ReadOnlyMany**: The volume can be mounted as read-only by many nodes.
   - **ReadWriteMany**: The volume can be mounted as read-write by many nodes.

16.9.6 PSQL Tab

The PSQL tab (figure 16.26) provides a simplified interface for interacting with the Zalando PostgreSQL Operator. It offers an overview of instances running on the cluster, it displays associated events, and provides credentials and access points for running PSQL databases. This information can be used later in Jupyter Notebooks or other PSQL clients.

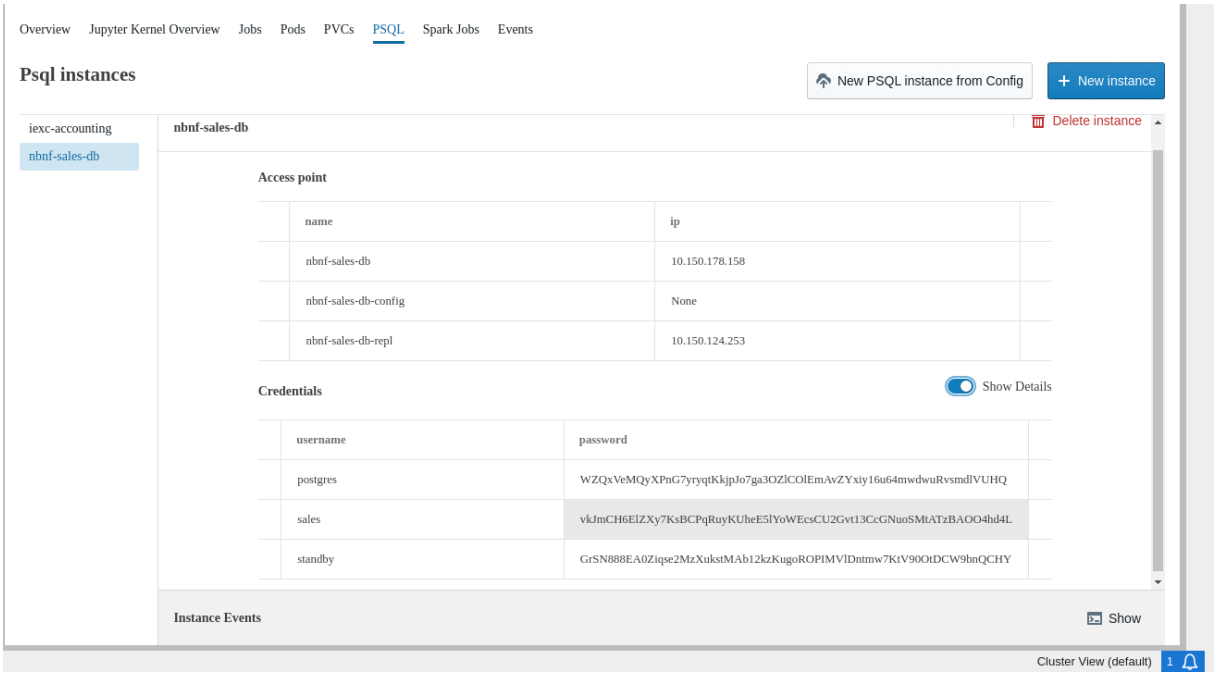


Figure 16.26: Jupyter Kubernetes Operators Manager: PSQL overview

The tab allows users to manage existing PostgreSQL databases or to create new ones via a dialog.

An example notebook is located at `/cm/shared/examples/jupyter/notebooks/psql-example.ipynb` (figure 16.27).

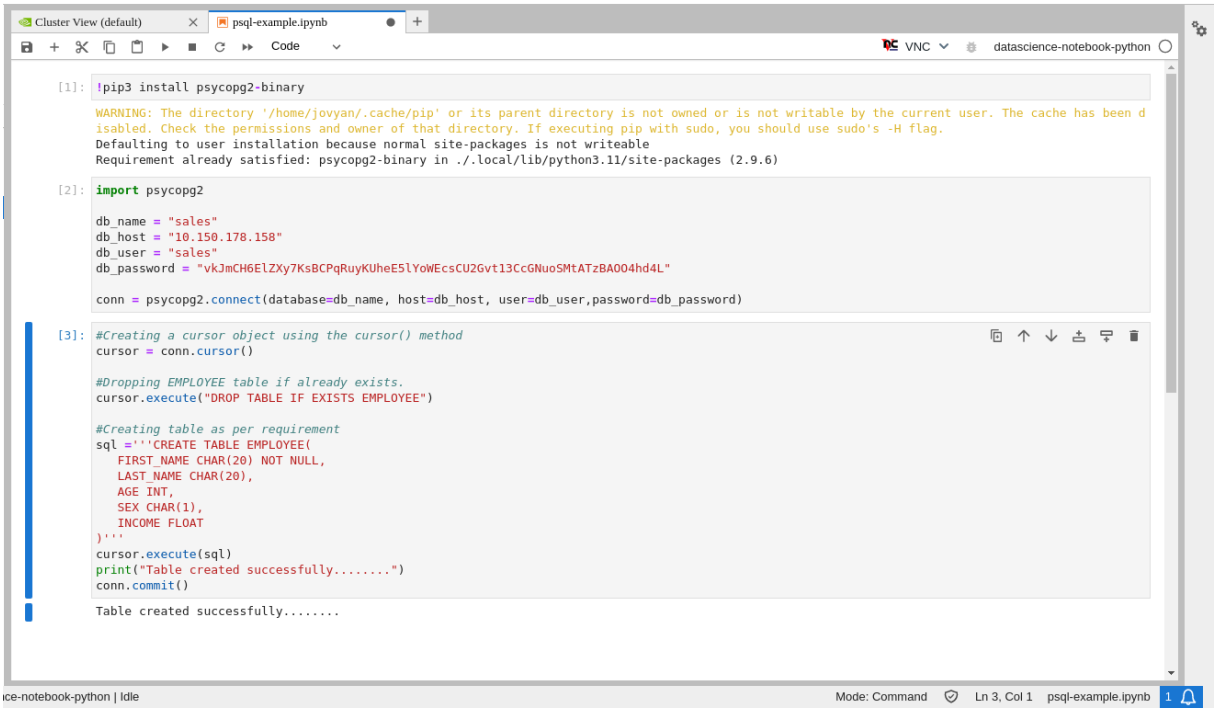


Figure 16.27: Jupyter Kubernetes Operators Manager: PSQL notebook

There are two buttons in figure 16.26:

- `New PSQL instance from Config`: allows a YAML file configuration that sets the fields to be submitted
- `New instance`: allows the fields to be set directly (figure 16.28)

Clicking either button displays a pop-up dialog for creating a new PostgreSQL instance. For example, the `New instance` button displays the creation form in figure 16.28.

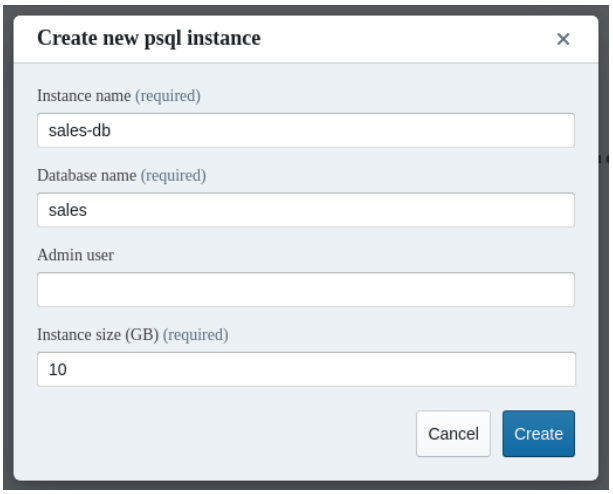


Figure 16.28: Jupyter Kubernetes Operators Manager: PSQL instance creation

The creation form accepts the following values:

1. **Instance name**: PSQL instance name. An auto-generated team name is prefixed to this name
2. **Database name**: arbitary database name

3. **Admin user** (Optional): if not set, the name of the admin user is set to the database name
4. **Instance size (GB)**: data storage size requirement for the volumes

16.9.7 Spark Tab

The `Spark Jobs` tab (figure 16.29) simplifies dealing with Spark clusters managed by the Google Spark Operator. The tab lets users list, remove, and create Spark instances using a simplified interface. If a particular option is not available in the simplified form, then a custom manifest can be uploaded.

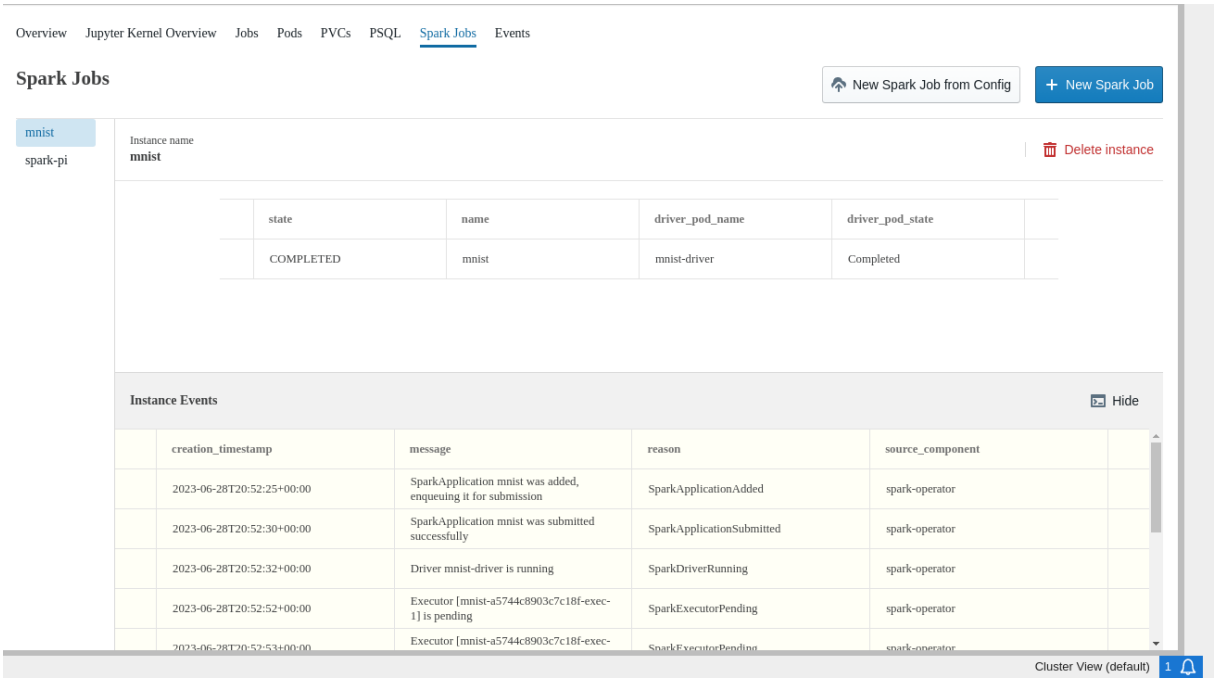


Figure 16.29: Jupyter Kubernetes Operators Manager: Spark overview

There are two buttons in figure 16.29:

- `New Spark Job from Config`: allows a YAML file configuration that sets the fields to be submitted
- `New Spark Job`: allows the fields to be set directly (figure 16.30)

Clicking either button displays a pop-up dialog for creating a new Spark instance. For example, the `New Spark Job` button displays the creation form in figure 16.30. The creation form requires the following values:

1. **Name**: Arbitrary name of the Spark instance.
2. **Image**: Container image for the driver and executor.
3. **Spark Version**: Version of Spark used by the application.
4. **Command**: Application's command with arguments.
5. **Executor Instances**: Number of executors.
6. **PVCs** (Optional): List of Persistent Volume Claims with mountpoints where these volumes are mounted. Used to store the application and datasets, as well as packed virtual environment archives.
7. **Virtual Environment** (Optional): Configuration for the 'spark.archives' configuration variable for the Python virtual environment.

- **PVC**: Name of the Persistent Volume Claim with empty Volume which are used to store the unpacked archive.
- **Archive**: Full path to the archive.
- **Python Path** (Optional): Python path (default `/bin/python`).

8. **Driver Resources** (Optional):
   - **Memory**: Amount of memory to request for the pod.
   - **GPU**: List of key-value pairs such as `nvidia.com/gpu, 3`.
   - **Cores**: Number of cores to use for the driver process.
   - **Environment**: List of key-value pairs for environment variables.
9. **Executor Resources** (Optional): Same as `Driver Resources`.
10. **Spark Config** (Optional): Custom configuration parameters if necessary. They are key-value pairs, documented in the Spark documentation at `https://spark.apache.org/docs/latest/configuration.html`.

Example: Calculating Pi

The simplest way to run Spark involves specifying just three values: the name, the script to execute, and the Spark version. For example, the container registry image `docker.io/apache/spark:3.5.1-python3` published by the Apache Spark project can be used, and an application to calculate the value for π is located at `/opt/spark/examples/src/main/python/pi.py` inside the image.

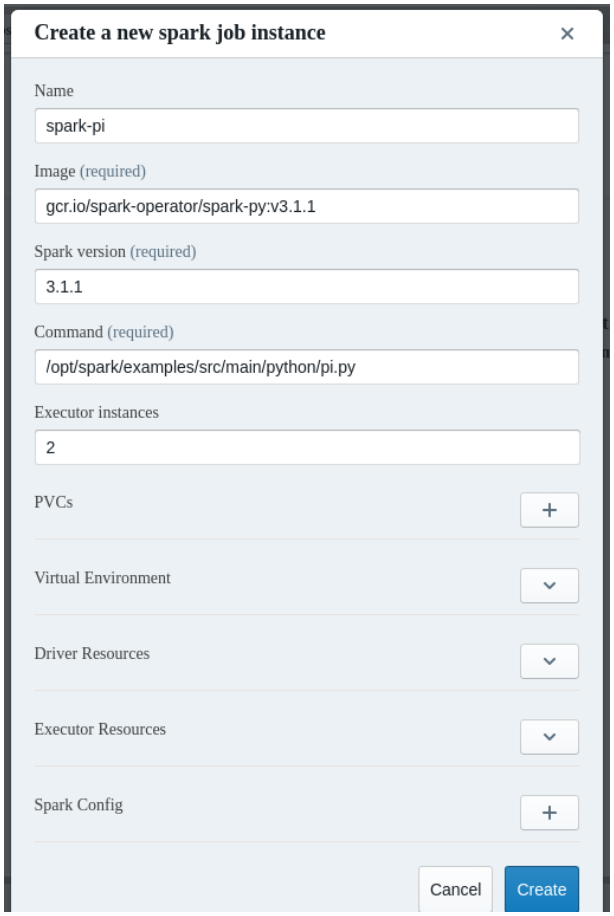


Figure 16.30: Jupyter Kubernetes Operators Manager: Spark creation

The progress of execution can be monitored in the `Pods` tab (figure 16.31):

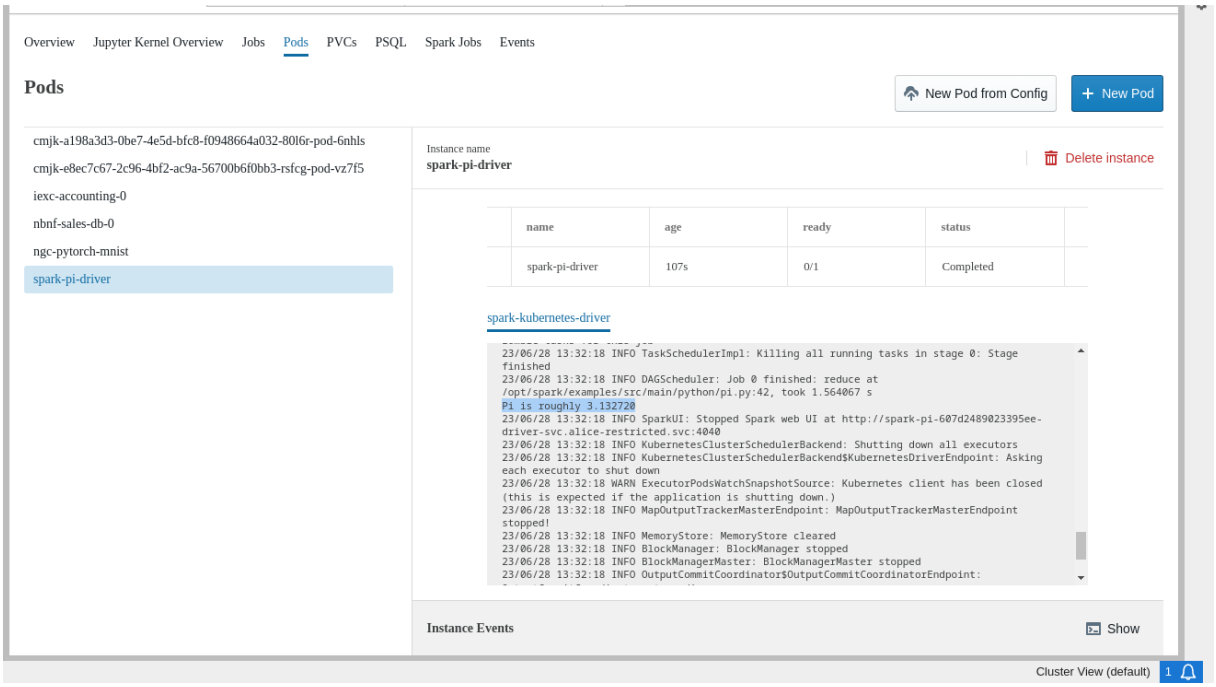


Figure 16.31: Jupyter Kubernetes Operators Manager: pods tab Spark example execution status

When the job is completed, the overview page is updated (figure 16.32):

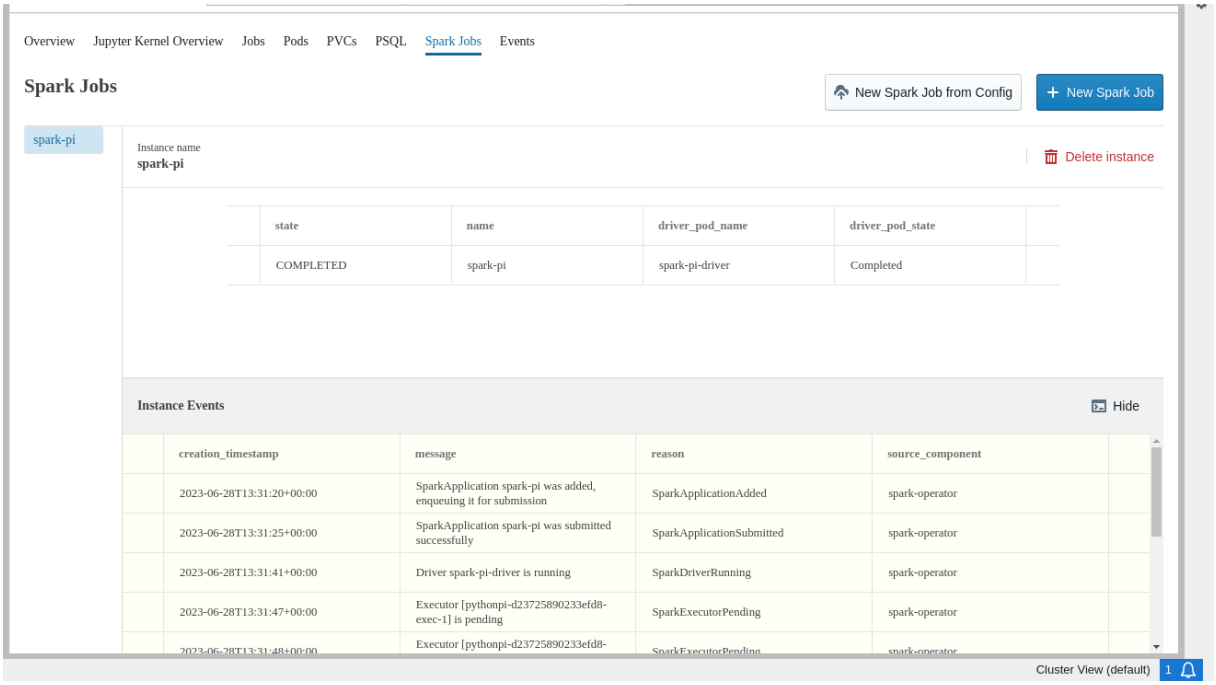


Figure 16.32: Jupyter Kubernetes Operators Manager: pods tab Spark example completed status

Example: Running MNIST

The source files for the MNIST example are located in the `/cm/shared/examples/jupyter/spark-operator-mnist` directory

Example

```
$ id
uid=1001(alice) gid=1001(alice) groups=1001(alice)
$ ls -l /home/alice/mnist/
total 12
-rwxr-xr-x 1 alice alice  241 Jun 28 16:02 create-venv.sh
drwxr-xr-x 2 alice alice   76 Jun 28 17:10 data
-rw-r--r-- 1 alice alice 2719 Jun 28 16:02 mnist.py
-rw-r--r-- 1 alice alice   46 Jun 28 16:02 requirements.txt
$ ls -l /home/alice/mnist/data
total 124952
-rw-rw-r-- 1 alice alice  18303650 Oct  1  2019 mnist_test.csv
-rw-rw-r-- 1 alice alice 109640201 Oct  1  2019 mnist_train.csv
-rw-r--r-- 1 alice alice        91 Jun 28 16:02 README.md
```

In the example the user is unable to train the model defined in `mnist.py` right away, as the image `docker.io/apache/spark:3.5.1-python3` does not have the required NumPy library. So, before running the Spark job, the virtual environment archive needs to be created.

The safest way to create the Python environment is to use the same context as where the archive is to be used later on. That way there are no missing or conflicting libraries and paths, and the Python version matches. The `docker.io/apache/spark:3.5.1-python3` image itself can therefore be used to create the archive. The script `create-venv.sh` creates the create archive from `requirements.txt` file and packs it.

In this example, a newly persistent volume (PV) is used as an intermediate storage for all the data and scripts, because using a file system where all the files are located would be a security concern. The reason for this is that once the hostPath is used for the pod, the running process UID/GID is dropped, and becomes the same as the original user (section 4.10.1 of the *Containerization Manual*). Spark images are usually not designed with this assumption in mind, and use UID 185 to run the application (`https://spark.apache.org/docs/3.1.1/running-on-kubernetes.html#user-identity`).

With these criteria in mind, the steps to run the MNIST example are:

1. Create a PVC for storing scripts and datasets (figure 16.33). Since the resulting persistent volume is expected to be mounted to all pods (driver and executors), its mode is set to `ReadWriteMany`.

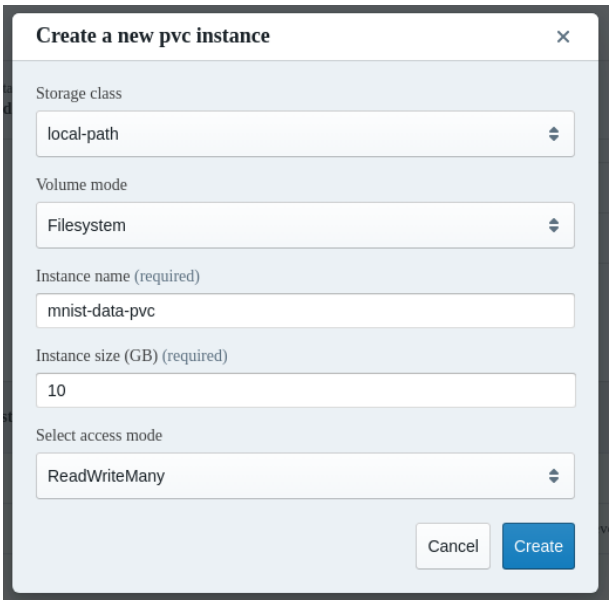


Figure 16.33: Jupyter Kubernetes Operators Manager: PVC creation for Spark example

2. Migrate data from the home folder to the volume. To do this, a migration job is created (figure 16.34), and allowed to finish (figure 16.35).

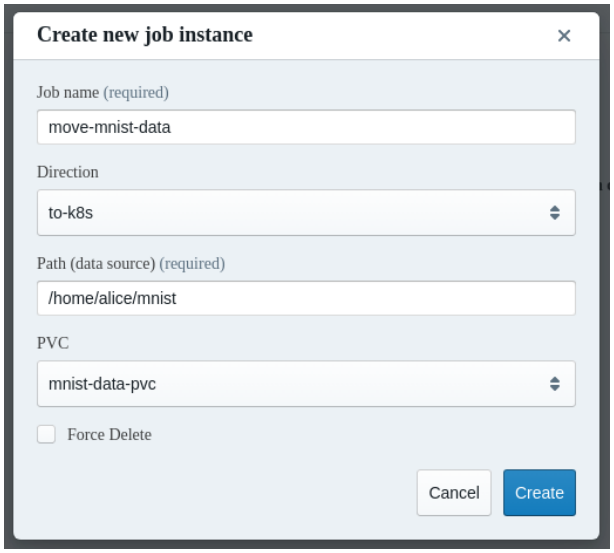


Figure 16.34: Jupyter Kubernetes Operators Manager: Data migration job creation for Spark example

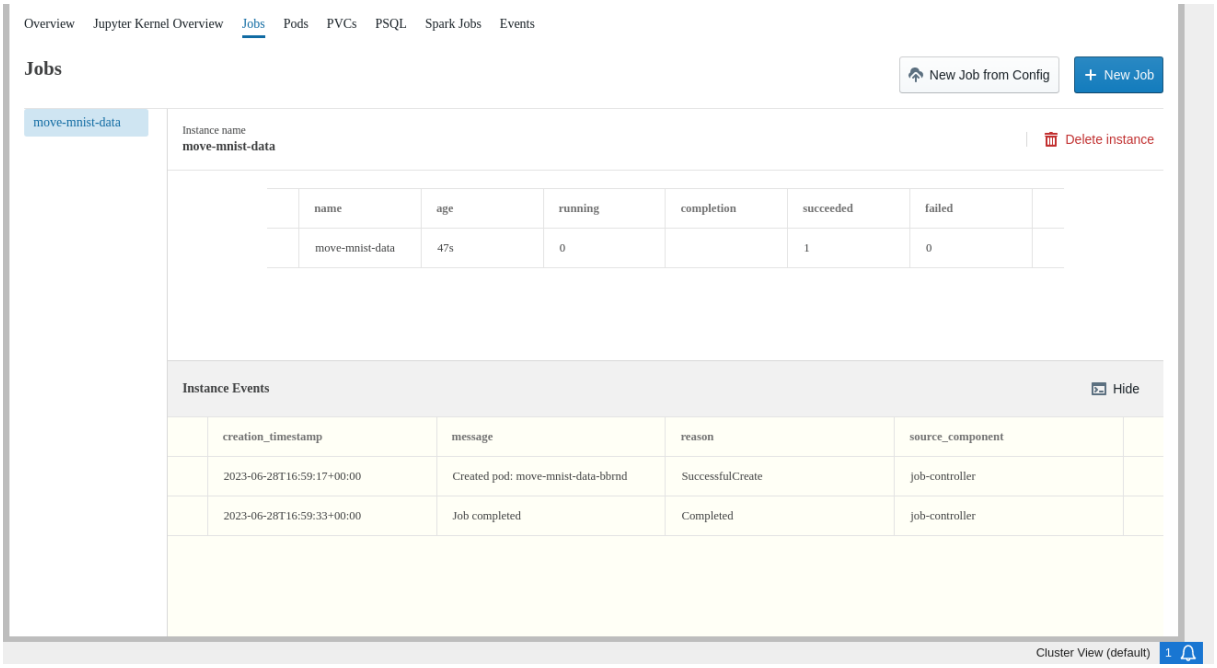


Figure 16.35: Jupyter Kubernetes Operators Manager: Data migration job for Spark example successfully finished

3. Create a pod to build a packed virtual environment (figure 16.36). For this, `mnist-data-pvc` is mounted by the user as the `/mnist` directory inside the pod. The command `/mnist/create-venv.sh /mnist/requirements.txt /mnist/venv.tar.gz` is then executed. This places the `venv.tar.gz` archive next to the other files in the volume.

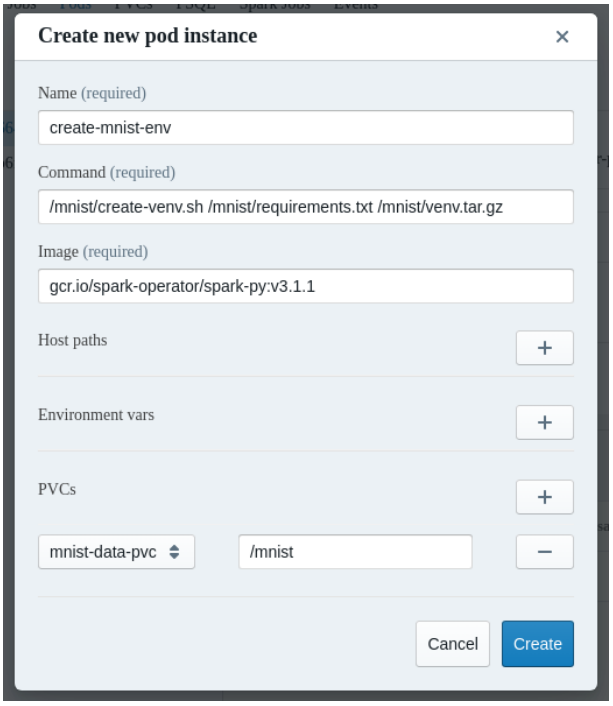


Figure 16.36: Jupyter Kubernetes Operators Manager: pod creation for virtual environment

4. (Optional) If necessary, data can be downloaded from the volume back to the user's home directory, to save the `venv.tar.gz` for future use (figure 16.37):

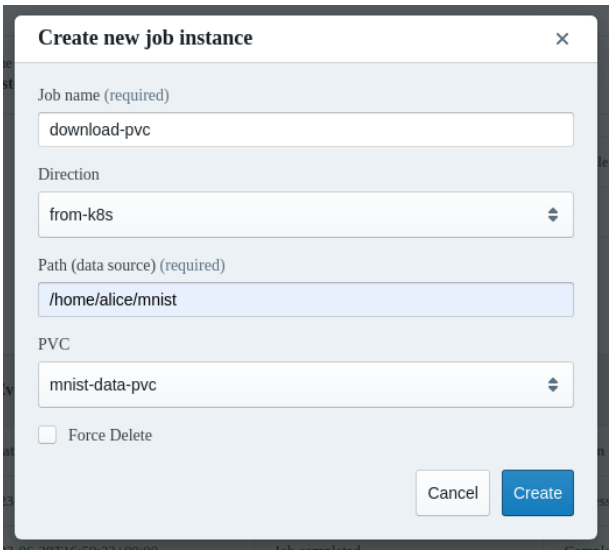


Figure 16.37: Jupyter Kubernetes Operators Manager: migrate venv to PVC

The `venv.tar.gz` file is placed in the `/home/alice/mnist` directory.

```
$ ls -l /home/alice/mnist/
total 261516
-rwxr-xr-x 1 alice alice       241 Jun 28 19:17 create-venv.sh
drwxr-xr-x 2 alice alice        99 Jun 28 19:17 data
-rw-r--r-- 1 alice alice      2719 Jun 28 19:17 mnist.py
-rw-r--r-- 1 alice alice        46 Jun 28 19:17 requirements.txt
```

```
-rw-r--r-- 1 alice alice 237896720 Jun 28 19:17 venv.tar.gz
```

5. Another volume is created to store the unpacked virtual environment files (figure 16.38). This is mounted to all pods (driver and executor). The driver unpacks the archive, and the content is used by the executors.

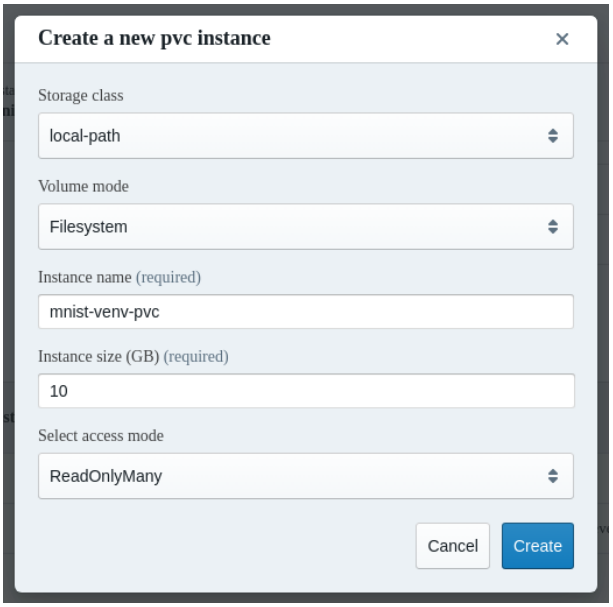


Figure 16.38: Jupyter Kubernetes Operators Manager: volume creation for unpacked virtual environment files

6. The Spark job can now be created (figure 16.39):

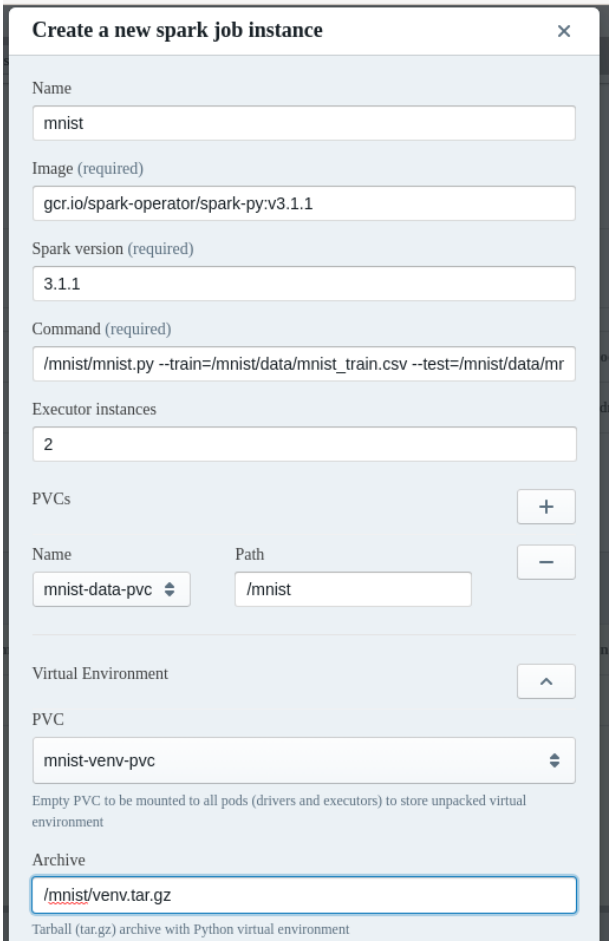


Figure 16.39: Jupyter Kubernetes Operators Manager: create Spark job

The execution progress can be monitored in the `Pods` tab (figure 16.40):

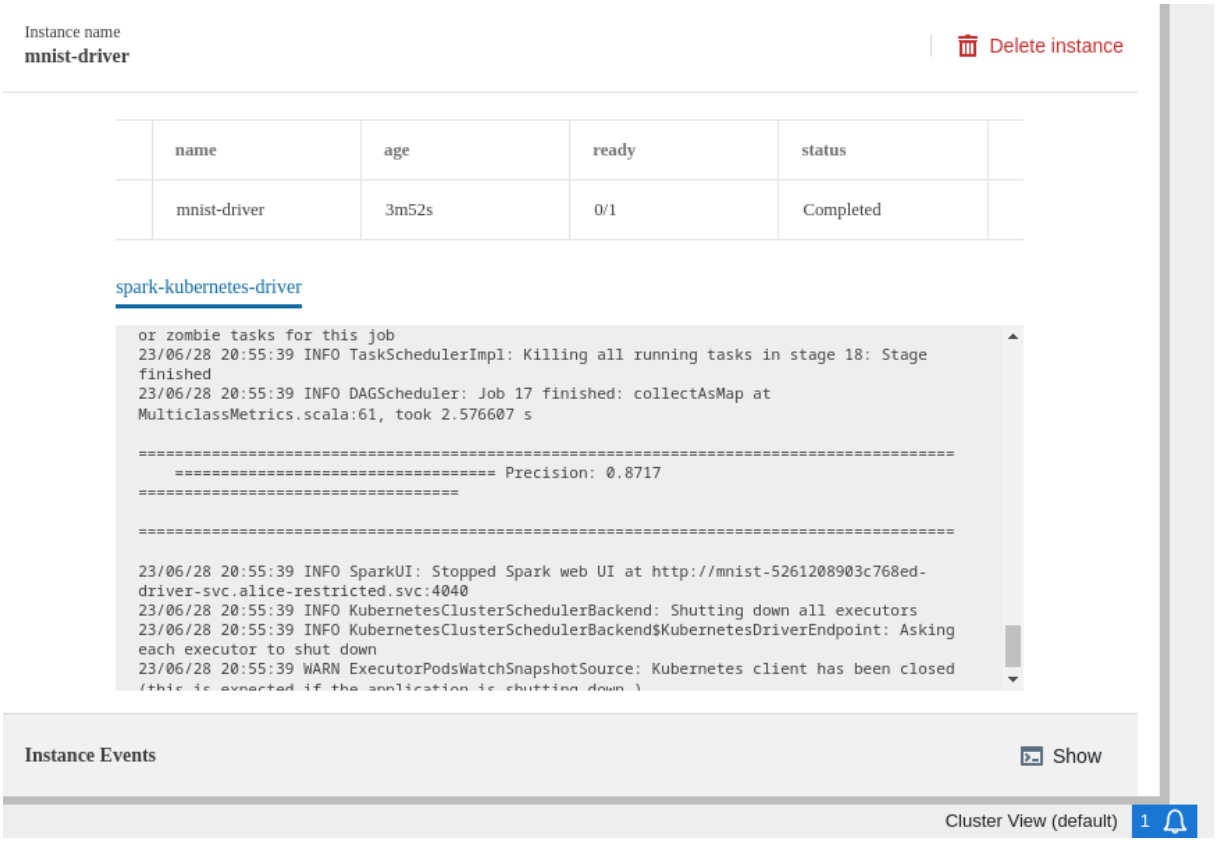


Figure 16.40: Jupyter Kubernetes Operators Manager: monitoring the Spark instance in the `Pods` tab

16.9.8 Events Tab

The `Events` tab lists all events from the user's namespace (figure 16.41):

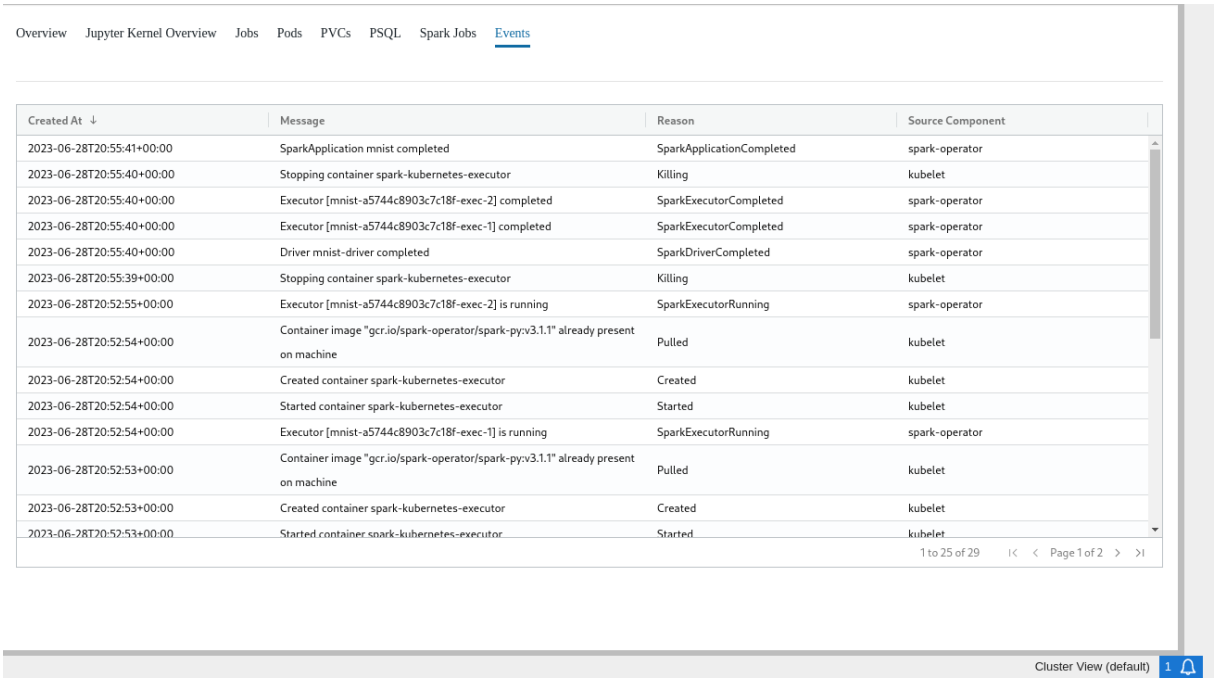


Figure 16.41: Jupyter Kubernetes Operators Manager: events overview

16.10 Jupyter Environment Removal

Before removing Jupyter, the administrator should ensure that all kernels have been halted, and that no user is still logged onto the web interface. Stopping the `cm-jupyterhub` service with users that are still logged in, or with running kernels, has undefined behavior.

To remove Jupyter, the script `cm-jupyter-setup` must be run, either in interactive mode, or with the option `--remove`.

Removing Jupyter does not remove or affect Kubernetes or WLM deployments.

For a more complete cleanup, the following packages must be manually removed from the nodes involved in the Jupyter deployment: `cm-jupyter`, `cm-jupyter-local`, and `cm-npm-configurable-http-proxy`.

A

Generated Files

This appendix contains lists of system configuration files that are managed by CMDaemon, and system configuration files that are managed by node-installer. These are files created or modified on the head nodes (section A.1), and on the regular nodes (sections A.2.3 and A.2.3). These files should not be confused with configuration files that are merely installed (section A.3).

Section 2.6.5 describes how system configuration files on all nodes are written out using the Cluster Management Daemon (CMDaemon). CMDaemon is introduced in section 2.6.5 and its configuration directives are listed in Appendix C.

All of these configuration files may be listed as `Frozen Files` in the CMDaemon configuration file to prevent them from being modified further by CMDaemon. The files can be frozen for the head node by setting the directive at `/cm/local/apps/cmd/etc/cmd.conf`. They can also be frozen on the regular nodes by setting the directive in the software image, by default at `/cm/images/default-image/cm/local/apps/cmd/etc/cmd.conf`.

A list of CMDaemon- and node-installer-managed files for nodes can be seen by running the command `filewriteinfo` in `device` mode. The command has options to run it for node groupings, as well useful path and sort options.

A.1 System Configuration Files Created Or Modified By CMDaemon On Head Nodes

If the `filewriteinfo` command is run from `device` mode, then it displays the last file written by CMDaemon or the node-installer on the nodes. Running the `filewriteinfo` command for the head node might display the following on a fresh installation:

```
[basecm11->device]% filewriteinfo basecm11
Hostname  Path                                                          Timestamp ... Actor Frozen
--------- ------------------------------------------------------------- --------- ... ----- ------
basecm11  /cm/images/default-image/etc/initramfs-tools/modules          Thu Jan 2 ... cmd   no
basecm11  /cm/images/default-image/etc/modprobe.d/bright-cmdaemon.conf  Thu Jan 2 ... cmd   no
basecm11  /cm/images/default-image/etc/securetty                        Thu Jan 2 ... cmd   no
basecm11  /cm/local/apps/openldap/etc/slapd.conf                        Thu Jan 2 ... cmd   no
basecm11  /etc/aliases                                                  Thu Jan 2 ... cmd   no
basecm11  /etc/bind/0.0.127.in-addr.arpa.zone                           Thu Jan 2 ... cmd   no
basecm11  /etc/bind/141.10.in-addr.arpa.zone                            Thu Jan 2 ... cmd   no
basecm11  /etc/bind/141.10.in-addr.arpa.zone.include                    Thu Jan 2 ... cmd   no
basecm11  /etc/bind/cm.cluster.zone                                     Thu Jan 2 ... cmd   no
basecm11  /etc/bind/cm.cluster.zone.include                             Thu Jan 2 ... cmd   no
basecm11  /etc/bind/eth.cluster.zone                                    Thu Jan 2 ... cmd   no
basecm11  /etc/bind/eth.cluster.zone.include                            Thu Jan 2 ... cmd   no
basecm11  /etc/bind/localhost.zone                                      Thu Jan 2 ... cmd   no
basecm11  /etc/bind/named.conf                                          Thu Jan 2 ... cmd   no
```

```
basecm11  /etc/bind/named.conf.include                                    Thu Jan 2 ... cmd   no
basecm11  /etc/default/nfs-kernel-server                                  Thu Jan 2 ... cmd   no
basecm11  /etc/dhcp/dhclient.conf                                         Thu Jan 2 ... cmd   no
basecm11  /etc/dhcpd.conf                                                 Thu Jan 2 ... cmd   no
basecm11  /etc/dhcpd.include.conf                                         Thu Jan 2 ... cmd   no
basecm11  /etc/dhcpd.internalnet.conf                                     Thu Jan 2 ... cmd   no
basecm11  /etc/dhcpd.internalnet.include.conf                             Thu Jan 2 ... cmd   no
basecm11  /etc/exports                                                    Thu Jan 2 ... cmd   no
basecm11  /etc/genders                                                    Thu Jan 2 ... cmd   no
basecm11  /etc/nslcd.conf                                                 Thu Jan 2 ... cmd   no
basecm11  /etc/ntpsec/ntp.conf                                            Thu Jan 2 ... cmd   no
basecm11  /etc/postfix/canonical                                          Thu Jan 2 ... cmd   no
basecm11  /etc/postfix/generic                                            Thu Jan 2 ... cmd   no
basecm11  /etc/postfix/main.cf                                            Thu Jan 2 ... cmd   no
basecm11  /etc/security/pam_bright.d/cm-check-alloc.conf                  Thu Jan 2 ... cmd   no
basecm11  /etc/shorewall.staging/interfaces                               Thu Jan 2 ... cmd   no
basecm11  /etc/shorewall.staging/netmap                                   Thu Jan 2 ... cmd   no
basecm11  /etc/shorewall.staging/policy                                   Thu Jan 2 ... cmd   no
basecm11  /etc/shorewall.staging/routes                                   Thu Jan 2 ... cmd   no
basecm11  /etc/shorewall.staging/snat                                     Thu Jan 2 ... cmd   no
basecm11  /etc/shorewall.staging/zones                                    Thu Jan 2 ... cmd   no
basecm11  /etc/sysconfig/dhcpd                                            Thu Jan 2 ... cmd   no
basecm11  /etc/systemd/resolved.conf                                      Thu Jan 2 ... cmd   no
basecm11  /tftpboot/images/default-image//boot                            Thu Jan 2 ... cmd   no
basecm11  /tftpboot/images/default-image/initrd                           Thu Jan 2 ... cmd   no
basecm11  /tftpboot/images/default-image/vmlinuz                          Thu Jan 2 ... cmd   no
basecm11  /tftpboot/mtu.conf                                              Thu Jan 2 ... cmd   no
basecm11  /tftpboot/pxelinux.cfg/category.default                         Thu Jan 2 ... cmd   no
basecm11  /tftpboot/pxelinux.cfg/default                                  Thu Jan 2 ... cmd   no
basecm11  /var/run/cmd.url                                                Thu Jan 2 ... cmd   no
basecm11  /var/run/my.cmd.url                                             Thu Jan 2 ... cmd   no
basecm11  /var/spool/cmd/config-head.json                                 Thu Jan 2 ... cmd   no
basecm11  /var/spool/cmd/my.cmd.url                                       Thu Jan 2 ... cmd   no
basecm11  /var/spool/cmd/my.master.cmd.url                                Thu Jan 2 ... cmd   no
basecm11  /var/spool/cmd/saved-config-files/cm-images-default-image-etc-
            initramfs-tools-modules                                       Thu Jan 2 ... cmd   no
basecm11  /var/spool/cmd/saved-config-files/cm-local-apps-openldap-etc-
            slapd.conf                                                    Thu Jan 2 ... cmd   no
basecm11  /var/spool/cmd/saved-config-files/etc-aliases                   Thu Jan 2 ... cmd   no
basecm11  /var/spool/cmd/saved-config-files/etc-bind-named.conf           Thu Jan 2 ... cmd   no
basecm11  /var/spool/cmd/saved-config-files/etc-dhcp-dhclient.conf        Thu Jan 2 ... cmd   no
basecm11  /var/spool/cmd/saved-config-files/etc-exports                   Thu Jan 2 ... cmd   no
basecm11  /var/spool/cmd/saved-config-files/etc-nslcd.conf                Thu Jan 2 ... cmd   no
basecm11  /var/spool/cmd/saved-config-files/etc-ntpsec-ntp.conf           Thu Jan 2 ... cmd   no
basecm11  /var/spool/cmd/saved-config-files/etc-postfix-main.cf           Thu Jan 2 ... cmd   no
basecm11  /var/spool/cmd/saved-config-files/etc-shorewall.staging-interfaces Thu Jan 2 ... cmd   no
basecm11  /var/spool/cmd/saved-config-files/etc-shorewall.staging-policy  Thu Jan 2 ... cmd   no
basecm11  /var/spool/cmd/saved-config-files/etc-shorewall.staging-snat    Thu Jan 2 ... cmd   no
basecm11  /var/spool/cmd/saved-config-files/etc-shorewall.staging-zones   Thu Jan 2 ... cmd   no
basecm11  /var/spool/cmd/saved-config-files/etc-sysconfig-dhcpd           Thu Jan 2 ... cmd   no
basecm11  /var/spool/cmd/saved-config-files/tftpboot-mtu.conf             Thu Jan 2 ... cmd   no
```

The `filewriteinfo` command is covered in more detail in section A.2.

The more important head node files that are managed by CMDaemon are listed here for a plain installation on the various distributions.

Some of the more important files managed automatically on the head node by CMDaemon In RHEL and derivatives, Ubuntu 22.04, 24.04, SLES15

File	Part	Comment
/cm/local/apps/openldap/etc/slapd.conf	Section	CMDaemon modifies this when edge, cloud or HA are activated
/cm/local/modulefiles/<module>	Entire directory	<module> is a file or directory for a modules environment module (section 2.2.4). For example, slurm, which is automatically created and made available for use after the Slurm workload manager is set up with cm-wlm-setup (section 7.3).
/cm/node-installer/etc/sysconfig/clock	Section	
/etc/aliases	Section	
/etc/bind/named.conf	Entire file	Ubuntu only. For zone additions use /etc/bind/named.conf.include[1]. For options additions, use /etc/bind/named.conf.global.options.include.
/etc/chrony.conf	Section	RHEL8,9 and derivatives only
/etc/dhcpd.conf	Entire file	
/etc/dhcp/dhclient.conf	Section	Not for SLES
/etc/dhcpd.internalnet.conf	Entire file	For internal networks other than internalnet, corresponding files are generated if node booting (table 3.1) is enabled
/etc/exports	Section	
/etc/genders	Section	
/etc/hosts	Section	
/etc/localtime	Symlink	
/etc/logrotate.d/slurm	Entire file	
/etc/logrotate.d/slurmdbd	Entire file	
/etc/named.conf	Entire file	Non-Ubuntu distributions only (Ubuntu uses /etc/bind/named.conf). For zone additions use /etc/named.conf.include[1]. For options additions, use /etc/named.conf.global.options.include.

...continues

...continued

File	Part	Comment
/etc/ntp.conf	Section	Ubuntu and SUSE only
/etc/postfix/canonical	Section	
/etc/postfix/generic	Section	
/etc/postfix/main.cf	Section	
/etc/resolv.conf	Section	The stub resolver can alternatively be provided by systemd-resolved (man systemd-resolved.8, page 97)
/etc/shorewall.staging/interfaces	Section	
/etc/shorewall.staging/netmap	Section	
/etc/shorewall.staging/policy	Section	
/etc/shorewall.staging/routes	Section	
/etc/shorewall.staging/snat	Section	
/etc/shorewall.staging/zones	Section	
/etc/snmp/snmptrapd.conf	Section	
/etc/sysconfig/bmccfg	Entire file	BMC configuration and BCM configuration
/etc/sysconfig/clock	Section	
/etc/sysconfig/dhcpd	Entire file	
/etc/sysconfig/network-scripts/ifcfg-*	Section	
/tftpboot/mtu.conf	Entire file	BCM configuration
/tftpboot/pxelinux.cfg/category.default	Entire file	BCM configuration
/tftpboot/pxelinux.cfg/default	Entire file	BCM configuration
/var/lib/named/*.zone[1,2]	Entire file	For SLES distributions only. For custom additions use /var/lib/named/*.zone.include
/var/named/*.zone[1,2]	Entire file	For RHEL8, RHEL9 distributions only. For custom additions use /var/named/*.zone.include

[1] User-added zone files ending in *.zone that are placed for a corresponding zone statement in the include file /etc/[bind/]named.conf.include are wiped by CMDaemon activity. Another pattern, eg: *.myzone, must therefore be used instead. Adding custom DNS zones with include files is described in section 3.2.3

...continues

...continued

File	Part	Comment

[2] For Ubuntu, the zone files are under `/etc/bind/`

A.2 System Configuration Files Created Or Modified Directly On The Node

Files that are created or modified by CMDaemon or the node-installer directly on the node by CMDaemon can be seen with the `filewriteinfo` command.

For example, for a particular node, in this case `node002` (some output elided):

Example

```
[basecm11->device]% filewriteinfo node002
Hostname    Path                                  Timestamp                 Actor          Frozen
----------- ------------------------------------- ------------------------- -------------- ------
node002     /certificates/fa-16-3e-54-bb-a1/cert  Thu Jul 14 12:16:00 2022  node-installer  no
...
node002     /etc/chrony.conf                      Thu Jul 14 12:17:08 2022  node-installer  no
...
node002     /etc/exports                          Thu Jul 14 12:17:58 2022  cmd             no
...
```

In the preceding, `/etc/exports` is seen to have been changed by `cmd`, which is listed under the `Actor` column.

A.2.1 Options To `filewriteinfo`

Options to `filewriteinfo` can be viewed with the `help filewwriteinfo` command.

Some of the options are:

- `-n|--node`: This can be used to specify a node list (page 75).
- `--field`: The `--field` option can be used with, for example, the `mac` field, as follows (some output elided):

  Example

  ```
  [basecm11->device]% get node002 mac
  FA:16:3E:54:BB:A1
  [basecm11->device]% filewriteinfo --field mac=FA:16:3E:54:BB:A1
  Hostname    Path                                  Timestamp                 Actor          Frozen
  ----------- ------------------------------------- ------------------------- -------------- ------
  node002     /certificates/fa-16-3e-54-bb-a1/cert  Thu Jul 14 12:16:00 2022  node-installer  no
  ...
  node002     /etc/chrony.conf                      Thu Jul 14 12:17:08 2022  node-installer  no
  ...
  node002     /etc/exports                          Thu Jul 14 12:17:58 2022  cmd             no
  ...
  ```

  The preceding example has the same output as for the session with `filewriteinfo node002` earlier.

  - A pitfall to avoid is the following: The columns headers of the output of the `filewriteinfo` command, such as `Actor`, are not keys for the `--field` option.

    Example

```
[basecm11->device]% filewriteinfo node002 --field Actor=cmd        #this is incorrect
```

Instead, as is the norm for a `device` mode command, it is the fields of the mode that are the key=value pairs (for keys that are valid for the device concerned). The fields of the `device` mode are displayed when the `format` command (page 62) is run. An example of a valid field would be `hostname=node001`.

Example

```
[basecm11->device]% filewriteinfo node002 --field hostname=node002  #this is correct
```

– Another pitfall with fields is that the `status` value, which is seen in the `format` output, is not actually a field—it is a dynamic property that requires an RPC.

Example

```
[basecm11->device]% filewriteinfo node002 --field status=UP        #this is incorrect
```

To filter on `status` for the node, the `-s|--status` option can be used.

Example

```
[basecm11->device]% filewriteinfo -s status up -n node002          #this is correct
```

- `--sort`: Sorts `filewriteinfo` output according to column headers (`Actor`, `Timestamp` and so on).
- `--path`: Restricts the display to files under the specified path. Multiple paths can be specified with comma-separated lists. If no path is specified, then there is no restriction.

Example

```
[basecm11->device]% filewriteinfo  --path /etc/chrony.conf --sort actor
Hostname         Path              Timestamp                 Actor           Frozen
---------------- ----------------- ------------------------- --------------- ------------
basecm11         /etc/chrony.conf  Thu Jul 14 12:14:25 2022  cmd             no
node001          /etc/chrony.conf  Thu Jul 14 12:17:08 2022  node-installer  no
node002          /etc/chrony.conf  Thu Jul 14 12:17:08 2022  node-installer  no
```

A.2.2 Files Created On Regular Nodes By CMDaemon

Files on a regular node that are modified by CMDaemon can be seen in the output of `filewriteinfo`. The `filewriteinfo` command displays a list of the last files written by the cluster manager.

For a default installation, the list is as shown in the following table:

System configuration files created or modified on a default regular nodes image by CMDaemon

File	Part	Comment
/etc/aliases	Section	
/etc/fstab	Section	SLES only
/etc/hosts	Section	
/etc/nslcd.conf	Section	RHEL8 and derivatives, and Ubuntu only
/etc/pam.d/sshd	Section	
/etc/postfix/main.cf	Section	
/etc/rsyslog.conf	Section	RHEL9 and derivatives, and SLES only
/etc/security/pam_bright.d/cm-check-alloc.conf	Entire file	
/var/run/cmd.url	Entire file	
/var/spool/cmd/my.cmd.url	Entire file	
/var/spool/cmd/my.master.cmd.url	Entire file	
/var/spool/cmd/saved-config-files/etc-aliases	Entire file Section	
/var/spool/cmd/saved-config-files/etc-fstab	Entire file	SLES Only
/var/spool/cmd/saved-config-files/etc-hosts	Entire file	
/var/spool/cmd/saved-config-files/etc-nslcd.conf	Entire file	RHEL8 and derivatives, and Ubuntu only
/var/spool/cmd/saved-config-files/etc-pam.d-sshd	Entire file	
/var/spool/cmd/saved-config-files/etc-postfix-main.cf	Entire file	
/var/spool/cmd/saved-config-files/etc-rsyslog.conf	Entire file	RHEL9 and derivatives, and SUSE only

A.2.3 Files Created On Regular Nodes By The Node-Installer

The list of files on a regular node that are modified by the node-installer during the last installation session can be viewed in the logs at /var/log/modified-by-node-installer.log on the node, and in the output of filewriteinfo. For a default installation, these are as shown in the following table:

System configuration files created or modified on regular nodes by the node-installer in RHEL8, RHEL9 and derivatives

File	Part	Comment
/etc/sysconfig/network-scripts/ifcfg-*	Entire	Interfaces for RHEL8, RHEL9 and derivatives only
/etc/network/interfaces.d/ifcfg-*	Entire file	Ubuntu only
/etc/sysconfig/network/ifcfg-*	Entire file	SLES only,
/etc/sysconfig/network	Section	Only for RHEL8, RHEL9 and derivatives
/etc/sysconfig/network/config	Section	Only for SLES
/etc/sysconfig/network/routes	Section	Only for SLES
/etc/resolv.conf	Entire file	Only for RHEL8, RHEL9 and derivatives
/etc/hosts	Section	
/etc/fstab	Section	Not for SLES. In SLES it is managed by CMDaemon
/etc/postfix/main.cf	Section	
/etc/chrony.conf	Entire file	RHEL8,RHEL9 and derivatives only, SLES
/etc/ntp.conf	Entire file	Ubuntu 22 only
/etc/ntpsec/ntp.conf	Entire file	Ubuntu 24.04 and later
/etc/systemd/resolved.conf	Entire file	Ubuntu only
/etc/hostname	Section	
/cm/local/apps/cmd/etc/cert.key	Section	
/cm/local/apps/cmd/etc/cert.pem	Section	
/cm/local/apps/cmd/etc/cluster.pem	Section	
/cm/local/apps/openldap/etc/certs/ldap.key	Section	
/cm/local/apps/openldap/etc/certs/ldap.pem	Section	
/var/log/modified-by-node-installer.log	Section	
/var/log/node-installer	Section	
/var/log/rsyncd.log	Section	
/var/spool/cmd/disks.xml	Section	

A.3 Files Not Generated, But Installed In RHEL And Derivatives

This appendix (Appendix A) is mainly about generated configuration files. This section (A.3) of the appendix discusses a class of files that is not generated, but may still be confused with generated files. The discussion in this section clarifies the issue, and explains how to check if non-generated installed files differ from the standard distribution installation.

A design goal of BCM is that of minimal interference. That is, to stay out of the way of the distributions that it works with as much as is reasonable. Still, there are inevitably cluster manager configuration files that are not generated, but installed from a cluster manager package. A cluster manager configuration file of this kind overwrites the distribution configuration file with its own special settings to get the cluster running, and the file is then not maintained by the node-installer or CMDaemon. Such files are therefore not listed on any of the tables in this chapter.

Sometimes the cluster file version may differ unexpectedly from the distribution version. To look into this, the following steps may be followed:

Is the configuration file a BCM version or a distribution version? A convenient way to check if a particular file is a cluster file version is to grep for it in the packages list for the cluster packages. For

example, for nsswitch.conf:

```
[root@basecm11 ~]# repoquery -l $(repoquery -a | grep -F _cmHEAD) | grep nsswitch.conf$
```

The inner repoquery displays a list of all the packages. By grepping for the cluster manager version string, for example _cmHEAD for NVIDIA Base Command Manager 11, the list of cluster manager packages is found. The outer repoquery displays the list of files within each package in the list of cluster manager packages. By grepping for nsswitch.conf$, any file paths ending in nsswitch.conf in the cluster manager packages are displayed. The output is:

```
/cm/conf/etc/nsswitch.conf
```

Files under /cm/conf are placed by BCM packages when updating the head node. From there they are copied over during the post-install section of the RPM to where the distribution version configuration files are located by the cluster manager, but only during the initial installation of the cluster. The distribution version file is overwritten in this way to prevent RPM dependency conflicts of the BCM version with the distribution version. The configuration files are not copied over from /cm/conf during subsequent reboots after the initial installation. The cm/conf files are however updated when BCM packages are updated. During such a BCM update, a notification is displayed that new configuration files are available.

Inverting the cluster manager version string match displays the files not provided by BCM. These are normally the files provided by the distribution:

```
[root@basecm11 ~]# repoquery -l $(repoquery -a | grep -F -v _cmHEAD) | grep nsswitch.conf$
...
/usr/share/factory/etc/nsswitch.conf
/usr/share/factory/etc/nsswitch.conf
/etc/authselect/nsswitch.conf
/etc/authselect/user-nsswitch.conf
/usr/share/authselect/default/minimal/nsswitch.conf
/usr/share/authselect/default/sssd/nsswitch.conf
/usr/share/authselect/default/winbind/nsswitch.conf
/var/lib/authselect/nsswitch.conf
/etc/authselect/nsswitch.conf
/etc/authselect/user-nsswitch.conf
/usr/share/authselect/default/minimal/nsswitch.conf
/usr/share/authselect/default/sssd/nsswitch.conf
/usr/share/authselect/default/winbind/nsswitch.conf
/var/lib/authselect/nsswitch.conf
/etc/nsswitch.conf
/etc/nsswitch.conf
/usr/share/authselect/vendor/libnss-mysql/nsswitch.conf
/usr/share/rear/skel/default/etc/nsswitch.conf
```

Which package provides the file in BCM and in the distribution? The packages that provide these files can be found by running the "yum whatprovides *" command on the paths given by the preceding output, for example:

```
~# yum whatprovides */cm/conf/etc/nsswitch.conf
...
cm-config-ldap-client<various types and versions are seen>
```

This reveals that some BCM LDAP packages can provide an nsswitch.conf file. The file is a plain file provided by the unpacking and placement that takes place when the package is installed. The file is not generated or maintained periodically after placement, which is the reason why this file is not seen in the tables of sections A.1 and A.2.3 of this appendix.

Similarly, looking through the output for the less specific case:

```
~# yum whatprovides */etc/nsswitch.conf
...
cm-config-ldap-client*
glibc*
rear*
systemd*
```

shows that `glibc` provides a distribution version of the `nsswitch.conf` file, that the `rear` package from the distribution provides a version of it, and that there is also a systemd version of this file available from the distribution packages. The glob `*` in the output in this manual represents a variety of types and versions. The actual display that is seen on the screen is an expansion of the glob.

Similar Ubuntu queries: For Ubuntu, a query of the form `dpkg -S` <*filename*> shows the packages that provide a file with the pattern <*filename*>.

Example

```
root@head:~# dpkg -S "nsswitch.conf"
cm-config-ldap-client-master: /cm/conf/etc/nsswitch.conf
manpages: /usr/share/man/man5/nsswitch.conf.5.gz
libc-bin: /usr/share/libc-bin/nsswitch.conf
```

To work out which Ubuntu package is a distribution package and which is a BCM package, queries similar to the following can be run:

Example

```
root@head:~# dpkg-query -W -f='${binary:Package} ${Version}\n' $(dpkg -S "nsswitch.conf" | cut -f1 -d:) |\
 grep cm10.0
cm-config-ldap-client-master 10.0-155-cm10.0
root@head:~# dpkg-query -W -f='${binary:Package} ${Version}\n' $(dpkg -S "nsswitch.conf" | cut -f1 -d:) |\
 grep -v cm10.0
libc-bin 2.35-0ubuntu3.8
manpages 5.10-1ubuntu1
```

What are the differences between the BCM version and the distribution versions of the file? Sometimes it is helpful to compare a distribution version and cluster version of `nsswitch.conf` to show the differences in configuration. The versions of the RPM packages containing the `nsswitch.conf` can be downloaded, their contents extracted, and their differences compared as follows:

```
~# mkdir yumextracted ; cd yumextracted
~# yumdownloader glibc-2.34
~# rpm2cpio glibc-2.34-100.el9_4.4.x86_64.rpm | cpio -idmv
~# yumdownloader cm-config-ldap-client-master
~# rpm2cpio cm-config-ldap-client-master-HEAD-155_cmHEAD.noarch.rpm | cpio -idmv
~# diff etc/nsswitch.conf cm/conf/etc/nsswitch.conf
...
```

What are the configuration files in an RPM package? An RPM package allows files within it to be marked as configuration files. Files marked as configuration files can be listed with `rpm -qc` <*package*>. Optionally, piping the list through "`sort -u`" filters out duplicates.

Example

```
~# rpm -qc glibc | sort -u
/etc/gai.conf
/etc/ld.so.cache
/etc/ld.so.conf
/etc/nsswitch.conf
/etc/rpc
/usr/lib64/gconv/gconv-modules
/usr/lib/gconv/gconv-modules
/var/cache/ldconfig/aux-cache
```

How does an RPM installation deal with local configuration changes? Are there configuration files or critical files that BCM misses? Whenever an RPM installation detects a file with local changes, it can treat the local system file as if:

1. the local system file is frozen. The installation does not interfere with the local file, but places the updated file as an .rpmnew file in the same directory.
2. the local system file is not frozen. The installation changes the local file. It copies the local file to an .rpmsave file in the same directory, and installs a new file from the RPM package.

NOTE: This package manager freezing should not be confused with CMDaemon freezing. With CMDaemon freezing, which is done with the `FrozenFile` directive (Appendix C), a file or section of a file that would normally be maintained by CMDaemon before freezing it, is no longer maintained by CMDaemon after freezing it.

When building BCM packages, the package builders can specify which of these two methods apply. When dealing with the built package, the system administrator can use an `rpm` query method to determine which of the two methods applies for a particular file in the package. For example, for `glibc`, the following query can be used and grepped:

```
rpm -q --queryformat '[%{FILENAMES}\t%{FILEFLAGS:fflags}\n]' glibc | egrep '[[:space:]].*(c|n).*$' | sort -u
/etc/gai.conf   cmng
/etc/ld.so.cache        cmng
/etc/ld.so.conf cn
/etc/nsswitch.conf      cn
/etc/rpc        cn
/usr/lib64/gconv/gconv-modules  cn
/usr/lib/gconv/gconv-modules    cn
/var/cache/ldconfig/aux-cache   cmng
```

Here, the second column of the output displayed shows which of the files in the package have a configuration (`c`) flag or a noreplace (`n`) flag. The `c` flag without the `n` flag indicates that an .rpmsave file will be created, while a `c` flag together with an `n` flag indicates that an .rpmnew file will be created.

In any case, files that are not marked as configuration files are overwritten during installation.

So:

- If a file is not marked as a configuration file, and it has been customized by the system administrator, and this file is provided by an RPM package, and the RPM package is updated on the system, then the file is overwritten silently.
- If a file is marked as a configuration file, and it has been customized by the system administrator, and this file is provided by an RPM package, and the RPM package is updated on the system, then it is good practice to look for .rpmsave and .rpmnew versions of that file, and run a comparison on detection.

BCM should however mark all critical files as configuration files in BCM packages.

Sometimes, RPM updates can overwrite a particular file that the administrator has changed locally and then would like to keep frozen.

To confirm that this is the problem, the following should be checked:

- The `--queryformat` option should be used to check that file can indeed be overwritten by updates. If the file has an `n` flag (regardless of whether it is a configuration file or not) then overwriting due to RPM updates does not happen, and the local file remains frozen. If the file has no `n` flag, then replacement occurs during RPM updates.

For files with no `n` flag, but where the administrator would still like to freeze the file during updates, the following can be considered:

- The file text content should be checked to see if it is a CMDaemon-maintained file (section 2.6.5), or checked against the list of generated files (Appendix A). This is just to make sure to avoid confusion about how changes are occurring in such a file.
  - If it is a CMDaemon-maintained file, then configuration changes put in by the administrator will also not persist in the maintained section of the file unless the `FrozenFile` directive (section C) is used to freeze the change.
  - If it is only a section that CMDaemon maintains, then configuration changes can be placed outside of the maintained section.

  Wherever the changes are placed in such a file, these changes are in any case by default overwritten on RPM updates if the file has no `n` flag.
- Some regular node updates can effectively be maintained in a desired state with the help of a finalize script (Appendix E).
- Updates can be excluded from YUM/zypper (section 9.3.2), thereby avoiding the overwriting of that file by the excluded package.

A request to change the package build flag may be sent to the BCM support team if the preceding suggested options are unworkable.

B

Bright Computing Public Key

BCM was a Bright Computing product originally, and many of its packages are still signed with the Bright Computing private key. The Bright Computing public key is published here as an independent reference.

```
-----BEGIN PGP PUBLIC KEY BLOCK-----
Version: GnuPG v1.4.0 (GNU/Linux)

mQGiBEqtYegRBADStdQjn1XxbYorXbFGncF2IcMFiNA7hamARt4w7hjtwZoKGHbC
zSLsQTmgZO+FZs+tXcZa5OLjGwhpxT6qhCe8Y7zIh2vwKrKlaAVKj2PUU28vKj1p
2W/OIiG/HKLtahLiCkOL3ahPOevJHh8B7elClrZOTKTBB6qIUbC5vHtjiwCgydm3
THLJsKnwk4qZetluTupldOEEANCzJ1nZxZzN6ZAMkIBrct8GivWClT1nBG4UwjHd
EDcGlREJxpg/OhpEP8TY1eOYUKRWvMqSVChPzkLUTIsd/O4RGTwOPGCo6Q3TLXpM
RVoonYPR1tRymPNZyW8VJeTUEnOkdlCaqZykp1sRb3jFAiJIRCmBRc854i/jRXmo
foTPBACJQyoEH9Qfe3VcqR6+vR2tX9lPvkxS7A5AnJIRs3Sv6yM4oV+7k/HrfYKt
fyl6widtEbQ187Os4x3NYXmmne7lz1nGxBfAxzPG9rtjRSXyVxc+KGVd6gKeCV6d
o7kS/LJHRiOLb5G4NZRFy5CGqg64liJwp/f2J4uyRbC8b+/LQbQ7QnJpZ2h0IENv
bXB1dGluZyBEZXZlbG9wbWVudCBUZWFtIDxkZXZAYnJpZ2h0Y29tcHV0aW5nLmNv
bT6IXgQTEQIAHgUCSq1h6AIbAwYLCQgHAwIDFQIDAxYCAQIeAQIXgAAKCRDvaS9m
+k3m0JO0AKC0GLTZiqoCQ6TRWW2ijjITEQ8CXACgg3o4oVbrG67VFzHUntcA0YTE
DXW5Ag0ESq1h6xAIAMJiaZI/0EqnrhSfiMsMT3sxz3mZkrQQL82Fob7s+S7nnMl8
A8btPzLlK8NzZytCglrIwPCYG6vfza/nkvyKEPh/f2it941bh7qiu4rBLqr+kGx3
zepSMRqIzW5FpIrUgDZOL9J+tWSSUtPW0YQ5jBBJrgJ8LQy9dK2RhAOLuHfbOSVB
JLIwNKxafkhMRwDoUNS4BiZKWyPFu47vd8fM67IPT1nMl0iCOR/QBn29MYuWnBcw
61344pd/IjOu3gM6YBqmRRU6yBeVi0TxxbYYnWcts6tEGAlTjHUOQ7gxVp4RDia2
jLVtbee8H464wxkkC3SSkng216RaBBAoaAykhzcAAwUH/iG4WsJHFw3+CRhUqy51
jnmb1FTFO8KQXI8JlPXMOh6vv0PtP5rw5D5V2cyVe2i4ez9Y8XMVfcbf60lptKyY
bRUjQq+9SNjt12ESU67YyLstSN68ach9Af03PoSZIKkiNwfA0+VBILv2Mhn7xd74
5LOM/eJ7lHSpeJA2Rzs6szc2340b/VxGfGWjogaK3NElSYOzQo+/kOVMdMWsQm/8
Ras19IA9P5jlSbcZQlHlPjndS4x4XQ8P41ATczsIDyWhsJC51rTuw9/QO7fqvvPn
xsRz1pFmiiN7I4JLjwOnAlXexn4EaeVa7Eb+uTjvxJZNdShs7Td74OmlF7RKFccI
wLuISQQYEQIACQUCSq1h6wIbDAAKCRDvaS9m+k3m0C/oAJsHMmKrLPhjCdZyHbB1
e19+5JABUwCfU0PoawBNOHzDnfr3MLaTgCwjsEE=
=WJX7
-----END PGP PUBLIC KEY BLOCK-----
```

C

CMDaemon Configuration File Directives

This appendix lists all configuration file directives that may be used in the cluster management daemon configuration file. If a change is needed, then the directives are normally changed on the head node, or on both head nodes in the high availability configuration, in:

`/cm/local/apps/cmd/etc/cmd.conf`

The directives can also be set in some cases for the regular nodes, via the software image in `/cm/images/default-image/cm/local/apps/cmd/etc/cmd.conf` on the head node. Changing the defaults already there is however not usually needed, and is not recommended.

Only one directive is valid per `cmd.conf` file.

To activate changes in a `cmd.conf` configuration file, the `cmd` service associated with it must be restarted.

- For the head node this is normally done with the command:

  ```
  systemctl restart cmd
  ```

- For regular nodes, `cmd` running on the nodes is restarted. Often, the image should be updated before `cmd` is restarted. How to carry out these procedures for a directive is described with an example where the FrozenFile directive is activated on a regular node on page 920.

Master directive

Syntax: `Master =` *hostname*
Default: `Master = master`

The cluster management daemon treats the host specified in the `Master` directive as the head node. A cluster management daemon running on a node specified as the head node starts in *head* mode. On a regular node, it starts in *node* mode.

Port directive

Syntax: `Port =` *number*
Default: `Port = 8080`

The *number* used in the syntax above is a number between 0 and 65535. The default value is 8080.

The `Port` directive sets the value of the port of the cluster management daemon to listen for non-SSL HTTP calls. By default, this happens only during init. All other communication with the cluster

management daemon is carried out over the SSL port. Pre-init port adjustment can be carried out in the node-installer.conf configuration. The head node firewall may need to be modified to allow traffic through for a changed port.

SSLPort directive

Syntax: `SSLPort = number`
Default: `SSLPort = 8081`

The *number* used in the syntax above is a number between 0 and 65535. The default value is 8081.

The `SSLPort` directive sets the value of the SSL port of the cluster management daemon to listen for SSL HTTP calls. By default, it is used for all communication of CMDaemon with Base View and `cmsh`, except for when CMDaemon is started up from init.

This directive does not change the firewall port settings to match the value of the SSL port for CMDaemon communication.

SSLPortOnly directive

Syntax: `SSLPortOnly = yes|no`
Default: `SSLPortOnly = no`

The `SSLPortOnly` directive allows the non-SSL port to be disabled. By default, during normal running, both SSL and non-SSL ports are listening, but only the SSL port is used. Also by default, the non-SSL port is only used during CMDaemon start up.

If `bootloaderprotocol` (section 5.1.6) is set to `HTTP`, then `SSLPortOnly` must be set to `no`. The HTTPS protocol is unsupported by most bootloaders.

Changing CMDaemon Ports

The `cm-cmd-ports` utility changes the HTTP and HTTPS ports on which CMDaemon listens. The default ports are 8080 (HTTP) and 8081 (HTTPS), as set by the `Port` (page 905) and `SSLPort` directives in `cmd.conf` (page 906).

The current port configuration can be listed with:

```
[root@basecm11 ~]# cm-cmd-ports -l
type              http    https  path
--------------  ------  -------  ------------------------------------------------------
image             8080     8081  /cm/images/default-image/cm/local/apps/cmd/etc/cmd.conf
image             8080     8081  /cm/local/apps/cmd/etc/cmd.conf
node-installer             8081  /cm/node-installer/scripts/node-installer.conf
```

To change the HTTPS port, for example to port 8082:

```
[root@basecm11 ~]# cm-cmd-ports --https 8082
Successfully updated ports
Changes take effect after restarting CMDaemon.
```

The firewall configuration is not automatically updated by `cm-cmd-ports`. The firewall rules file must be updated manually as part of the procedure. This procedure should be performed during a maintenance window.

Example

An example session is as follows:

On the active head node, the port is changed with:

```
[root@basecm11 ~]# cm-cmd-ports --https 8082
```

The software images are updated on the provisioners with the `updateprovisioners` command (page 250), and an update is triggered for all non-image fsparts with the `trigger` command (page 252):

```
[basecm11]% softwareimage updateprovisioners
...
[basecm11]% fspart; trigger -a
...
```

The provisioning status can be checked with:

```
[basecm11]% softwareimage provisioningstatus -r
```

An empty output indicates that provisioning is complete, and that the procedure may continue. CMDaemon can then be restarted on the active head node:

```
[root@basecm11 ~]# systemctl restart cmd
```

In a high availability configuration, CMDaemon on the passive head node is restarted as well, but only after the `CMDaemon` on the active head node has successfully restarted. This order keeps the passive in sync with the active. Updating the passive first can cause issues later on.

The updated `cmd.conf` is then copied from the software images to the compute nodes:

```
[root@basecm11 ~]# pdcp -w <comma-separated node list> -p
/cm/images/<image name>/cm/local/apps/cmd/etc/cmd.conf /cm/local/apps/cmd/etc/cmd.conf
```

The command is run for each node image in use on the cluster.

CMDaemon on the compute nodes is then restarted:

```
[root@basecm11 ~]# pdsh -g computenode systemctl restart cmd
```

The firewall rules on the head nodes are updated by editing `/etc/shorewall/rules` to replace the old port number with the new port number, after which Shorewall is reloaded:

```
[root@basecm11 ~]# shorewall reload
```

The port change can be verified with:

```
[root@basecm11 ~]# cm-cmd-ports -l
type              http    https  path
--------------  ------  -------  ------------------------------------------------------
image             8080     8082  /cm/images/default-image/cm/local/apps/cmd/etc/cmd.conf
image             8080     8082  /cm/local/apps/cmd/etc/cmd.conf
node-installer            8082  /cm/node-installer/scripts/node-installer.conf
[root@basecm11 ~]# cmsh -c ds
basecm11 ............................ [   UP   ]
node001 ............................. [   UP   ]
```

From BCM 12 onwards, Shorewall is superseded by nftables.

CertificateFile directive

Syntax: `CertificateFile =` *filename*
Default: `CertificateFile = "/cm/local/apps/cmd/etc/cert.pem"`

The `CertificateFile` directive specifies the PEM-format certificate which is to be used for authentication purposes. On the head node, the certificate used also serves as a software license.

PrivateKeyFile directive

Syntax: `PrivateKeyFile = filename`
Default: `PrivateKeyFile = "/cm/local/apps/cmd/etc/cert.key"`

The `PrivateKeyFile` directive specifies the PEM-format private key which corresponds to the certificate that is being used.

CACertificateFile directive

Syntax: `CACertificateFile = filename`
Default: `CACertificateFile = "/cm/local/apps/cmd/etc/cacert.pem"`

The `CACertificateFile` directive specifies the path to the BCM PEM-format root certificate. It is normally not necessary to change the root certificate.

ClusterCertificateFile directive

Syntax: `ClusterCertificateFile = filename`
Default: `ClusterCertificateFile = "/cm/local/apps/cmd/etc/cluster.pem"`

The `ClusterCertificateFile` directive specifies the path to the BCM PEM-format cluster certificate file used as a software license, and to sign all client certificates

ClusterPrivateKeyFile directive

Syntax: `ClusterPrivateKeyFile = filename`
Default: `ClusterPrivateKeyFile = "/cm/local/apps/cmd/etc/cluster.key"`

The `ClusterPrivateKeyFile` directive specifies the path to the BCM PEM-format private key which corresponds to the cluster certificate file.

RandomSeedFile directive

Syntax: `RandomSeedFile = filename`
Default: `RandomSeedFile = "/dev/urandom"`

The `RandomSeedFile` directive specifies the path to a source of randomness for a random seed.

RandomSeedFileSize directive

Syntax: `RandomSeedFileSize = number`
Default: `RandomSeedFileSize = 8192`

The `RandomSeedFileSize` directive specifies the size of the random seed.

DHParamFile directive

Syntax: `DHParamFile = filename`
Default: `DHParamFile = "/cm/local/apps/cmd/etc/dh1024.pem"`

The `DHParamFile` directive specifies the path to the Diffie-Hellman parameters.

SSLHandshakeTimeout directive

Syntax: `SSLHandshakeTimeout = number`
Default: `SSLHandshakeTimeout = 10`

The `SSLHandShakeTimeout` directive controls the time-out period (in seconds) for SSL handshakes.

SSLSessionCacheExpirationTime directive

Syntax: `SSLSessionCacheExpirationTime = number`
Default: `SSLSessionCacheExpirationTime = 300`

The `SSLSessionCacheExpirationTime` directive controls the period (in seconds) for which SSL sessions are cached. Specifying the value `0` can be used to disable SSL session caching.

DBHost directive

Syntax: `DBHost = hostname`
Default: `DBHost = "localhost"`

The `DBHost` directive specifies the hostname of the MySQL database server.

DBPort directive

Syntax: `DBPort = number`
Default: `DBPort = 3306`

The `DBPort` directive specifies the TCP port of the MySQL database server.

DBUser directive

Syntax: `DBUser = username`
Default: `DBUser = cmdaemon`

The `DBUser` directive specifies the username used to connect to the MySQL database server.

DBPass directive

Syntax: `DBPass = password`
Default: `DBPass = "<random string set during installation>"`

The `DBPass` directive specifies the password used to connect to the MySQL database server.

DBName directive

Syntax: `DBName = database`
Default: `DBName = "cmdaemon"`

The `DBName` directive specifies the database used on the MySQL database server to store CMDaemon related configuration and status information.

DBUnixSocket directive

Syntax: `DBUnixSocket = filename`

Default: `DBUnixSocket = "/var/lib/mysql/mysql.sock"`

The `DBUnixSocket` directive specifies the named pipe used to connect to the MySQL database server if it is running on the same machine.

DBUpdateFile directive

Syntax: `DBUpdateFile =` *filename*
Default: `DBUpdateFile = "/cm/local/apps/cmd/etc/cmdaemon_upgrade.sql"`

The `DBUpdateFile` directive specifies the path to the file that contains information on how to upgrade the database from one revision to another.

EventBucket directive

Syntax: `EventBucket =` *filename*
Default: `EventBucket = "/var/spool/cmd/eventbucket"`

The `EventBucket` directive (section 10.10.3) specifies the path to the named pipe that is created to listen for incoming events from a user.

EventBucketFilter directive

Syntax: `EventBucketFilter =` *filename*
Default: `EventBucketFilter = "/cm/local/apps/cmd/etc/eventbucket.filter"`

The `EventBucketFilter` directive (section 10.10.3) specifies the path to the file that contains regular expressions used to filter out incoming messages on the event-bucket.

LDAPHost directive

Syntax: `LDAPHost =` *hostname*
Default: `LDAPHost = "localhost"`

The `LDAPHost` directive specifies the hostname of the LDAP server to connect to for user management.

LDAPUser directive

Syntax: `LDAPUser =` *username*
Default: `LDAPUser = "root"`

The `LDAPUser` directive specifies the username used when connecting to the LDAP server.

LDAPPass directive

Syntax: `LDAPPass =` *password*
Default: `LDAPPass = "<random string set during installation>"`

The `LDAPPass` directive specifies the password used when connecting to the LDAP server. It can be changed following the procedure described in Appendix I.

LDAPReadOnlyUser directive

Syntax: `LDAPReadOnlyUser =` *username*
Default: `LDAPReadOnlyUser = "readonlyroot"`

The `LDAPReadOnlyUser` directive specifies the username that will be used when connecting to the LDAP server during LDAP replication. The user is a member of the "rogroup" group, whose members have a read-only access to the whole LDAP directory.

LDAPReadOnlyPass directive

Syntax: `LDAPReadOnlyPass =` *password*
Default: `LDAPReadOnlyPass = "<random string set during installation>"`

The `LDAPReadOnlyPass` directive specifies the password that will be used when connecting to the LDAP server during LDAP replication.

LDAPSearchDN directive

Syntax: `LDAPSearchDN =` *dn*
Default: `LDAPSearchDN = "dc=cm,dc=cluster"`

The `LDAPSearchDN` directive specifies the Distinguished Name (DN) used when querying the LDAP server.

LDAPProtocol directive

Syntax: `LDAPProtocol = ldap|ldaps`
Default: `LDAPProtocol = "ldaps"`

The `LDAPProtocol` directive specifies the LDAP protocol to be used when querying the LDAP server.

LDAPPort directive

Syntax: `LDAPPort =` *number*
Default: `LDAPPort = 636`

The `LDAPPort` directive specifies the port to be used when querying the LDAP server.

LDAPCACertificate directive

Syntax: `LDAPCACertificate =` *filename*
Default: `LDAPCACertificate = "/cm/local/apps/openldap/etc/certs/ca.pem"`

The `LDAPCACertificate` directive specifies the CA certificate to be used when querying the LDAP server.

LDAPCertificate directive

Syntax: `LDAPCertificate =` *filename*
Default: `LDAPCertificate = "/cm/local/apps/openldap/etc/certs/ldap.pem"`

The `LDAPCertificate` directive specifies the LDAP certificate to be used when querying the LDAP server.

LDAPPrivateKey directive

Syntax: `LDAPPrivateKey=` *filename*
Default: `LDAPPrivateKey= "/cm/local/apps/openldap/etc/certs/ldap.key"`

The `LDAPPrivateKey` directive specifies the LDAP key to be used when querying the LDAP server.

HomeRoot directive

Syntax: `HomeRoot =` *path*
Default: `HomeRoot = "/home"`

The `HomeRoot` directive specifies the default user home directory used by CMDaemon. It is used for automatic mounts, exports, and when creating new users.

DocumentRoot directive

Syntax: `DocumentRoot =` *path*
Default: `DocumentRoot = "/cm/local/apps/cmd/etc/htdocs"`

The `DocumentRoot` directive specifies the directory mapped to the web-root of the CMDaemon. The CMDaemon acts as a HTTP-server, and can therefore in principle also be accessed by web-browsers.

SpoolDir directive

Syntax: `SpoolDir =` *path*
Default: `SpoolDir = "/var/spool/cmd"`

The `SpoolDir` directive specifies the directory which is used by the CMDaemon to store temporary and semi-temporary files.

EnableShellService directive

Syntax: `EnableShellService = true|false`
Default: `EnableShellService = true`

The `EnableShellService` directive allows shells to be started from Base View.

The connection runs over CMDaemon, which is running over SSL, which means that between Base View and the device, the connection is encrypted.

The directive does not affect `cmsh`'s `rshell`, `rconsole` `telnet`, and `ssh` commands.

DisableRemoteShell directive

Syntax: `AdvancedConfig = {"DisableRemoteShell=0|1", ...}`
Default: `DisableRemoteShell=0`
`DisableRemoteShell` is a parameter of the `AdvancedConfig` (page 925) directive.

By default CMDaemon provides access to devices via `rshell`, `rconsole` (SOL), `telnet`, and `ssh`. Setting the directive to `1` disables all RPC shell access.

A more fine-grained disabling is possible with the following `AdvancedConfig` directives, which follow the same syntax:

- **DisableRemoteRShell directive**: disables a remote shell to the head node.
- **DisableRemoteRShellImage directive**: disables a remote shell to the chrooted software image.

- **DisableRemoteRShellNode directive**: disables a remote shell to the regular nodes.
- **DisableRemoteRShellTelnet directive**: disables a telnet shell to a device.
- **DisableRemoteRShellSSH directive**: disables SSH to a device.
- **DisableRemoteSOL directive**: disables a console via SOL to a device.

EnableWebSocketService directive

Syntax: `EnableWebSocketService = true|false`
Default: `EnableWebSocketService = true`

The `EnableWebSocketService` directive allows the use of CMDaemon Lite (section 2.6.7).

EnablePrometheusMetricService directive

Syntax: `EnablePrometheusMetricService = true|false`
Default: `EnablePrometheusMetricService = true`

If true, the `EnablePrometheusMetricService` directive creates an HTTP endpoint for Prometheus-style exporters that do not have their own HTTP endpoint.

PrometheusMetricServicePath directive

Syntax: `PrometheusMetricServicePath =` *path*
Default: `PrometheusMetricServicePath = SCRIPTS_DIR"metrics/prometheus"`

The `PrometheusMetricServicePath` directive is the path from which CMDaemon can serve Prometheus metrics. `SCRIPTS_DIR` is the stem path `/cm/local/apps/cmd/scripts` by default.

EnablePrometheusExporterService directive

Syntax: `EnablePrometheusExporterService = true|false`
Default: `EnablePrometheusExporterService = false`

If true, the `EnablePrometheusExporterService` directive allows a Prometheus server to pull data from CMDaemon.

To collect all data, the external Prometheus YAML configuration (`https://prometheus.io/docs/prometheus/latest/configuration/configuration/`) should have all nodes with a monitoring role added to it:

Example

```
#
# - job_name: cmdaemon-head01
#   metrics_path: /exporter
#   static_configs:
#     - targets: ['https://head01:8081']
#
```

Configuring CMDaemon to sample metrics from a Prometheus exporter instead is covered in section 3.18.

PrometheusExporterRequireCertificate directive

Syntax: `AdvancedConfig = {"PrometheusExporterRequireCertificate=0|1", ...}`
Default: `PrometheusExporterRequireCertificate = 1`

`PrometheusExporterRequireCertificate` is a parameter of the `AdvancedConfig` (page 925) directive.

- If set to `0`, then all users can use the Prometheus service to pull data using CMDaemon.
- If set to `1`, then only users using certificate-based access can use the Prometheus service to pull data using CMDaemon.

Certificate-based access can be configured as follows:
A custom profile with the name `prometheus` can be created for BCM users with `profile` mode (section 6.5) as follows:

```
[basecm11->profile]% add prometheus
[basecm11->profile*[prometheus*]]% append tokens PROMETHEUS_EXPORTER_TOKEN
[basecm11->profile*[prometheus*]]% set nonuser yes
[basecm11->profile*[prometheus*]]% commit
```

A 2048-bit SSL certificate called `mypromcert` can be created for a non-privileged user `prometheus` with 3650 days validity using the `createcertificate` command (section 6.5.2) as follows:

```
[basecm11->profile[prometheus]]% cert
[basecm11->cert]% createcertificate 2048 mypromcert "" "" "" "" "" prometheus "" 3650 /tmp/prometheus.key  \
                  /tmp/prometheus.pem
Certificate key written to file: /tmp/prometheus.key
Certificate pem written to file: /tmp/prometheus.pem
```

The PEM certificate is the public part of the key pair. The private key must be owned and be readable by the Prometheus instance that is scraping from the node exporter.
To configure SSL authentication between Prometheus server and the Prometheus exporter node:

- On the Prometheus server:
  - A directive value of:
    - `0` means that the session key is a temporary key that is auto-generated for any user
    - `1` means that the user (the Prometheus server) must provide a certificate
  - CMDaemon in a default cluster manager installation uses a self-signed certificate. A copy of certificate can be picked up on the head node with:

    Example

    ```
    [root@basecm11 ~]# ca_cert="/etc/prometheus/cmdaemon-$(hostname)-ca.pem"
    [root@basecm11 ~]# cat /cm/local/apps/cmd/etc/{cacert,cluster}.pem > ${ca_cert}
    [root@basecm11 ~]# chmod 600 ${ca_cert}
    ```

    The Prometheus server should be configured to specify this CA certificate.
    If the certificate used by CMDaemon is not a self-signed certificate—that is to say, it is signed by a recognized CA—then the certificate issued by CMDaemon is legitimate, and the Prometheus server does not need to have that CA certificate explicitly specified.
- The Prometheus exporter always uses TLS, which is automatically managed by CMDaemon.

EnablePrometheusPushService directive

Syntax: `EnablePrometheusPushService = true|false`
Default: `EnablePrometheusPushService = false`

If true, the `EnablePrometheusPushService` directive allows a Prometheus server to push data to CMDaemon:

Example

```
myhead # cat <<EOF > "$file"
myhead # test_prometheus_push{hello="world"} 42
myhead # EOF
myhead # curl -k -X POST --data-binary "@$file" https://localhost:8081/prometheuspush/job/test
```

CMDaemonAudit directive

Syntax: `CMDaemonAudit = true|false`
Default: `CMDaemonAudit = true`

When the `CMDaemonAudit` directive is set to `true`, and a value is set for the CMDaemon auditor file with the `CMDaemonAuditorFile` directive, then CMDaemon actions are time-stamped and logged in the CMDaemon auditor file.

CMDaemonAuditorFile directive

Syntax: `CMDaemonAuditorFile =` *path to* `audit.log` *file*
or
Syntax: `CMDaemonAuditorFile =` *path to* `audit.json` *file*
or
Syntax: `CMDaemonAuditorFile =` *http(s) path to JSON server*

Default: `CMDaemonAuditorFile = "/var/spool/cmd/audit.log"`

The `CMDaemonAuditorFile` directive sets where the audit logs for CMDaemon actions are logged. The log format for a `.log` file in a standard directory path is:
(*time stamp*) *profile* [*IP-address*] *action* (*unique key*)

Example

```
(Mon Jan 31 12:41:37 2011) Administrator [127.0.0.1] added Profile: arbitprof(4294967301)
```

The directive can be set in the following kinds of formats:

- `CMDaemonAuditorFile = "/var/spool/cmd/audit.log"`
- `CMDaemonAuditorFile = "/var/spool/cmd/audit.json"`
- `CMDaemonAuditorFile = "http://<IP address>:<port>/some/path"`
- `CMDaemonAuditorFile = "https://<IP address>:<port>/some/path"`

A simple POST web service can be faked using `netcat`:

Example

```
nc -l 1234 -k
```

The JSON file always contains a valid array. An RPC call looks like this:

Example

```
...
{
  "entity": "node001",
  "rpc": {
    "address": "127.0.0.1",
    "call": "updateDevice",
    "service": "cmdevice",
    "timestamp": 1579185696,
    "user": "Administrator.root"
  },
  "task_id": 0,
  "updated": true
},
...
```

DisableAuditorForProfiles directive

Syntax: `DisableAuditorForProfiles = {`*profile* `[, `*profile*`] ...}`
Default: `DisableAuditorForProfiles = {NODE}`

The `DisableAuditorForProfiles` directive sets the profile for which an audit log for CMDaemon actions is disabled. A profile (section 2.3.4) is what allows or denies what CMDaemon may do for that profile user. More than one profile can be set as a comma-separated list. New profiles can also be created via the `profile` mode of `cmsh` or via the navigation path `Identity Management > Profiles >` of Base View, thus making it possible to disable auditing for arbitrary groups of CMDaemon services.

EventLogger directive

Syntax: `EventLogger = true|false`
Default: `EventLogger = true`

The `EventLogger` directive sets whether to log events. If active, then by default it logs events to `/var/spool/cmd/events.log` on the active head. If a failover takes place, then the event logs on both heads should be checked and merged for a complete list of events.

The location of the event log on the filesystem can be changed using the `EventLoggerFile` directive (page 916).

Whether events are logged in files or not, events are cached and accessible using `cmsh` or Base View. The number of events cached by CMDaemon is determined by the parameter `MaxEventHistory` (page 916).

EventLoggerFile directive

Syntax: `EventLoggerFile =` *filename*
Default: `EventLoggerFile = "/var/spool/cmd/events.log"`

The `EventLogger` directive sets where the events seen in the event viewer (section 10.10) are logged.

MaxEventHistory directive

Syntax: `AdvancedConfig = {"MaxEventHistory=`*number*`", ...}`

Default: `MaxEventHistory=8192`
`MaxEventHistory` is a parameter of the `AdvancedConfig` (page 925) directive.

By default, when not explicitly set, the maximum number of events that is retained by CMDaemon is 8192. Older events are discarded.

The parameter can take a value from 0 to 1000000. However, CMDaemon is less responsive with larger values, so in that case, setting the `EventLogger` directive (page 916) to `true`, to activate logging to a file, is advised instead.

TimingOverview directive

Syntax: `TimingOverview =` *filename*
Default: `TimingOverview = true|false`

If set to `true`, the `TimingOverview` directive records timing information for CMDaemon.

TimingOverviewFile directive

Syntax: `TimingOverviewFile =` *filename*
Default: `TimingOverviewFile = "/var/spool/cmd/timing.overview.log"`

The `TimingOverviewFile` directive sets the file where the timing data values for CMDaemon go.

PublicDNS directive

Syntax: `PublicDNS = true|false`
Default: `PublicDNS = false`

By default, internal hosts are resolved only if requests are from the internal network. Setting `PublicDNS` to true allows the head node name server to resolve internal network hosts for any network, including networks that are on other interfaces on the head node. Separate from this directive, port 53/UDP must also be opened up in Shorewall (section 7.2 of the *Installation Manual*) if DNS is to be provided for queries from an external network.

MaximalSearchDomains directive

Syntax: `GlobalConfig = {"MaximalSearchDomains =` *number*`", ...}`
Default: `64`

The `MaximalSearchDomains` directive is a parameter of the `GlobalConfig` (page 926) directive. It sets the maximum number of search domains that BCM configures for the cluster, and can be set to any value from 1 to 1024. The default was 6 before NVIDIA Base Command Manager version 10.23.09, 32 from version 10.23.09 onward, and 64 from point release 11.35.0 onward.

On older versions of glibc (before 2.26), only the first 6 configured search domains are used by the operating system, regardless of this directive's setting. For example, to raise the limit to 100 domains on a cluster running glibc 2.26 or later, the directive setting would be: `GlobalConfig = { "MaximalSearchDomains=100" }`

LockDownDhcpd directive

Syntax: `LockDownDhcpd = true|false`
Default: `LockDownDhcpd = false`

LockDownDhcpd is a deprecated legacy directive. If set to true, a global DHCP "deny unknown-clients" option is set. This means no new DHCP leases are granted to unknown clients for all networks. Unknown clients are nodes for which BCM has no MAC addresses associated with the node. The directive LockDownDhcpd is deprecated because its globality affects clients on all networks managed by BCM, which is contrary to the general principle of segregating the network activity of networks.

The recommended way now to deny letting new nodes boot up is to set the option for specific networks by using cmsh or Base View (section 3.2.1: figure 3.5 and table 3.1). Setting the cmd.conf LockDownDhcpd directive overrides lockdowndhcpd values set by cmsh or Base View.

MaxNumberOfProvisioningThreads directive

Syntax: MaxNumberOfProvisioningThreads = *number*
Default: MaxNumberOfProvisioningThreads = 10000

The MaxNumberOfProvisioningThreads directive specifies the cluster-wide total number of nodes that can be provisioned simultaneously. Individual provisioning servers typically define a much lower bound on the number of nodes that may be provisioned simultaneously.

SetupBMC directive

Syntax: SetupBMC = true|false
Default: SetupBMC = true

Automatically configure the username and password for the BMC interface of the head node. This may also be valid for regular nodes. The SetupBMC directive should not be confused with the setupBmc field of the node-installer configuration file, described in section 5.8.8.

The node-installer normally takes care of BMC interface configuration on regular nodes by acting on the node-installer configuration file field setupBmc. The CMDaemon directive SetupBMC can only work on regular nodes if the node-installer is not configuring the regular nodes. If the boolean parameters installbootrecord and allownetworkingrestart for a regular node are set to yes, then the SetupBMC directive is able to work for that regular node. Setting the boolean parameters at category level also makes the directive work for the associated nodes.

BMCSessionTimeout directive

Syntax: BMCSessionTimeout = *number*
Default: BMCSessionTimeout = 2000

The BMCSessionTimeout specifies the time-out for BMC calls in milliseconds.

BMCIdentifyScript directive

Syntax: AdvancedConfig = {"BMCIdentify=*filename*", ...}
Default: *unset*
BMCIdentifyScript is a parameter of the AdvancedConfig (page 925) directive.

The parameter takes a full file path to a script that can be used for identification with a BMC (section 3.7.5).

BMCIdentifyScriptTimeout directive

Syntax: AdvancedConfig = {"BMCIdentifyScriptTimeout=*number from 1 to 360*", ...}
Default: 60
BMCIdentifyScriptTimeout is a parameter of the AdvancedConfig (page 925) directive.

CMDaemon waits at the most BMCIdentifyScriptTimeout seconds for the script used by the BMCIdentify directive to complete.

BMCIdentifyCache directive

Syntax: AdvancedConfig = {"BMCIdentifyCache=0|1", ...}
Default: 1
BMCIdentifyCache is a parameter of the AdvancedConfig (page 925) directive.

If set to 1, then CMDaemon remembers the last value of the output of the script used by the BMCIdentify directive.

SnmpSessionTimeout directive

Syntax: SnmpSessionTimeout = *number*
Default: SnmpSessionTimeout = 500000

The SnmpSessionTimeout specifies the time-out for SNMP calls in microseconds.

PowerOffPDUOutlet directive

Syntax: PowerOffPDUOutlet = true|false
Default: PowerOffPDUOutlet = false

Enabling the PowerOffPDUOutlet directive allows PDU ports to be powered off for clusters that have both PDU and IPMI power control. Section 4.1.3 has more on this.

PowerThreadPoolSize directive

Syntax: AdvancedConfig = {"PowerThreadPoolSize=<*integer*>", ...}
Default: PowerThreadPoolSize=32

PowerThreadPoolSize is a parameter of the AdvancedConfig (page 925) directive.

The parameter can take positive integer values. Increasing its value increases the number of threads that are used to power up the nodes in a cluster (section 4.2.3), so that the cluster is fully operational quicker. The administrator should however take into account the power surge due to increasing the number of threads (number of subprocesses) before increasing the value beyond its default.

DisableBootLogo directive

Syntax: DisableBootLogo = true|false
Default: DisableBootLogo = false

When DisableBootLogo is set to true, the BCM logo is not displayed on the first boot menu when nodes PXE boot.

StoreBIOSTimeInUTC directive

Syntax: StoreBIOSTimeInUTC = true|false
Default: StoreBIOSTimeInUTC = false

When StoreBIOSTimeInUTC is set to true, the system relies on the time being stored in BIOS as being UTC rather than local time.

FreezeChangesTo<*wlm*>Config directives: FreezeChangesToSlurmConfig directive FreezeChangesToLSFConfig directive

Syntax: `FreezeChangesTo<wlm>Config= true|false`
Default: `FreezeChangesTo<wlm>Config = false`

When `FreezeChangesTo<wlm>Config` is set to `true`, the CMDaemon running on that node does not make any modifications to the workload manager configuration for that node. Workload managers for which this value can be set are:

- Slurm
- LSF

Monitoring of jobs, and workload accounting and reporting continues for frozen workload managers.

Upgrades to newer workload manager versions may still require some manual adjustments of the configuration file, typically if a newer version of the workload manager configuration changes the syntax of one of the options in the file.

FrozenFile directive

Syntax: `FrozenFile = { filename[, filename]... }`
Example: `FrozenFile = { "/etc/dhcpd.conf", "/etc/postfix/main.cf" }`
The `FrozenFile` directive is used to prevent the CMDaemon-maintained sections of configuration files from being automatically generated. This is useful when site-specific modifications to configuration files have to be made.

The `filewriteinfo` command (section A.2) can be run from `device` mode to see which files are frozen. The output in the `Frozen` column then shows which files are protected from CMDaemon-managed changes.

To avoid problems, the file that is frozen should not be a symlink, but should be the ultimate destination file. The `readlink -f <symlinkname>` command returns the ultimate destination file of a symlink called <*symlinkname*>. This is also the case for an ultimate destination file that is reached via several chained symlinks.

FrozenFile directive for regular nodes

FrozenFile directive for regular nodes for CMDaemon

The `FrozenFile` directive can be used within the `cmd.conf` file of the regular node.

Example

To freeze the file `/etc/named.conf` on the regular nodes running with the image `default-image`, the file:

```
/cm/images/default-image/cm/local/apps/cmd/etc/cmd.conf
```

can have the following directive set in it:

```
FrozenFile = { "/etc/named.conf" }
```

The path of the file that is to be frozen on the regular node must be specified relative to the root of the regular node.

The running node should then have its image updated. This can be done with the `imageupdate` command in `cmsh` (section 5.6.2), or the `Update node` button in Base View (section 5.6.3). After the update, CMDaemon should be restarted within that category of nodes:

Example

```
[root@basecm11 ~]# pdsh -v -g category=default systemctl restart cmd
node002: Waiting for CMDaemon (25129) to terminate...
node001: Waiting for CMDaemon (19645) to terminate...
node002: [  OK  ]
node001: [  OK  ]
node002: Waiting for CMDaemon to start...[  OK  ]
node001: Waiting for CMDaemon to start...[  OK  ]
```

FrozenFile regex specification

The `FrozenFile` directive allows regexes to be used for a path, if the path begins with the `|` character:

Example

```
[root@head current]# egrep -e '^FrozenFile' /cm/local/apps/cmd/etc/cmd.conf
FrozenFile = { "/etc/postfix/main.cf", "|/cm/images/.*?/etc/postfix/main.cf" }
```

In the preceding entry, all image directories under `/cm/images/` are matched for the path `etc/postfix/main.cf`.

FrozenFile directive for regular nodes for the node-installer

CMDaemon directives only affect files on a regular node after CMDaemon starts up on the node during the init stage. So files frozen by the CMDaemon directive stay unchanged by CMDaemon after this stage, but they may still be changed before this stage.

Freezing files so that they also stay unchanged during the pre-init stage—that is during the node-installer stage—is possible with node-installer directives.

Node-installer freezing is independent of CMDaemon freezing, which means that if a file freeze is needed for the entire startup process as well as beyond, then both a node-installer as well as a CMDaemon freeze are sometimes needed.

Node-installer freezes can be done with the node-installer directives in `/cm/node-installer/scripts/node-installer.conf`, introduced in section 5.4:

- `frozenFilesPerNode`
- `frozenFilesPerCategory`

For the `node-installer.conf` file in multidistro and multiarch (section 9.7) configurations, the directory path `/cm/node-installer` takes the form:

`/cm/node-installer-`<*distribution*>`-`<*architecture*>

The values for <*distribution*> and <*architecture*> can take the values outlined on page 558.

Example

Per node:

```
frozenFilesPerNode = { "*:/localdisk/etc/ntp.conf", "node003:/localdisk/etc/hosts" }
```

Here, the `*` wildcard means that no restriction is set. Setting `node003` means that only node003 is frozen.

Example

Per category:

```
frozenFilesPerCategory = { "mycategory:/localdisk/etc/sysconfig/network-scripts/ifcfg-eth1" }
```

Here, the nodes in the category `mycategory` are prevented from being changed by the node-installer.

Is A FrozenFile Directive Always Needed?

A FrozenFile directive is not always needed because in some cases a workaround is possible.

In a configuration file after a node is fully up, the effect of a statement earlier on can often be overridden by a statement later in the file. So, the following useful behavior is independent of whether FrozenFile is being used for a configuration file or not: A configuration file, for example /etc/postfix/main.cf, with a configuration statement in an earlier CMDaemon-maintained part of the file, for example:

```
mydomain = eth.cluster
```

can often be overridden by a statement later on outside the CMDaemon-maintained part of the file:

```
mydomain = eth.gig.cluster
```

Using FrozenFile in CMDaemon or the node-installer can thus sometimes be avoided by the use of such overriding statements later on.

Whether overriding later on is possible depends on the software being configured. It is true for Postfix configuration files, for example, but it may not be so for the configuration files of other applications.

EaseNetworkValidation directive

Syntax: EaseNetworkValidation = 0|1|2
Default: EaseNetworkValidation = 0

CMDaemon enforces certain requirements on network interfaces and management/node-booting networks by default. In heavily customized setups, such as is common in Type 3 networks (section 3.3.9 of the *Installation Manual*), the user may wish to disable these requirements.

- 0 enforces all requirements.
- 1 allows violation of the requirements, with validation warnings. This value should never be set except under instructions from BCM support.
- 2 allows violation of the requirements, without validation warnings. This value should never be set except under instructions from BCM support.

CustomUpdateConfigFileScript directive

Syntax: CustomUpdateConfigFileScript = *filename*
Default: *commented out in the default* cmd.conf *file*

Whenever one or more entities have changed, the custom script at *filename*, specified by a full path, is called 30s later. Python bindings can be used to get information on the current setup.

ConfigDumpPath directive

Syntax: ConfigDumpPath = *filename*
Default: ConfigDumpPath = /var/spool/cmd/cmdaemon.config.dump

The ConfigDumpPath directive sets a dump file for dumping the configuration used by the power control script /cm/local/apps/cmd/scripts/pctl/pctl. The pctl script is a fallback script to allow power operations if CMDaemon is not running.

- If no directive is set (ConfigDumpPath = ""), then no dump is done.

- If a directive is set, then the administrator must match the variable `cmdconfigfile` in the powercontrol configuration file `/cm/local/apps/cmd/scripts/pctl/config.py` to the value of `ConfigDumpPath`. By default, the value of `cmdconfigfile` is set to `/var/spool/cmd/cmdaemon.config.dump`.

SyslogHost directive

Syntax: `SyslogHost =` *hostname*
Default: `SyslogHost = "localhost"`

The `SyslogHost` directive specifies the hostname of the syslog host.

SyslogFacility directive

Syntax: `SyslogFacility =` *facility*
Default: `SyslogFacility = "LOG_LOCAL6"`

The default value of `LOG_LOCAL6` is set in:

- `/etc/rsyslog.conf` in RHEL, Ubuntu.
- `/etc/syslog-ng/syslog-ng.conf` in SLES versions

These are the configuration files for the default syslog daemons `syslog`, `rsyslog`, and `syslog-ng`, respectively, that come with the distribution. BCM redirects messages from CMDaemon to `/var/log/cmdaemon` only for the default syslog daemon that the distribution provides. So, if another syslog daemon other than the default is used, then the administrator has to configure the non-default syslog daemon facilities manually.

The value of *facility* must be one of: `LOG_KERN`, `LOG_USER`, `LOG_MAIL`, `LOG_DAEMON`, `LOG_AUTH`, `LOG_SYSLOG` or `LOG_LOCAL0..7`

NameServerLocalhostLocation directive

Syntax: `AdvancedConfig = {"NameServerLocalhostLocation=0|1", ...}`
Default: `NameServerLocalhostLocation=0`

`NameServerLocalhostLocation` is a parameter of the `AdvancedConfig` (page 925) directive.

When set to `1`, the location of the localhost as specified by the nameserver directive in `/etc/resolv.conf` is moved to the bottom of the list of nameserver entries. The default value of `0` places it at the top of those entries.

ResolveToExternalName directive

Syntax: `ResolveToExternalName = true|false`
Default: `ResolveToExternalName = false`

The value of the `ResolveToExternalName` directive determines under which domain name the primary and secondary head node hostnames are visible from within the head nodes, and to which IP addresses their hostnames are resolved. Enabling this directive resolves the head nodes' hostnames to the IP addresses of their external interfaces.

Thus, on head nodes and regular nodes in both single-head and failover clusters

- with `ResolveToExternalName` disabled, the `master` hostname and the actual hostname of the head node (e.g. `head1`, `head2`) by default always resolve to the internal IP address of the head node.

- with ResolveToExternalName enabled, the master hostname and the actual hostname of the head node (e.g. head1, head2) by default always resolve to the external IP address of the head node.

The resolution behavior can be summarized by the following table:

ResolveToExternalName Directive Effects

on simple head, resolving:		on failover head, resolving:			on regular node, resolving:		Using the
master	head	master	head1	head2	master	head(s)	DNS?
ResolveToExternalName = False							
I	I	I	I	I	I	I	No
I	I	I	I	I	I	I	Yes
ResolveToExternalName = True							
E	E	E	E	E	E	E	No
E	E	E	E	E	E	E	Yes

Key: I: resolves to internal IP address of head
E: resolves to external IP address of head

The system configuration files on the head nodes that get affected by this directive include /etc/hosts and, on SLES systems, also the /etc/HOSTNAME. Also, the DNS zone configuration files get affected.

Additionally, in both the single-head and failover clusters, using the "hostname -f" command on a head node while ResolveToExternalName is enabled results in the host's Fully Qualified Domain Name (FQDN) being returned with the host's external domain name. That is, the domain name of the network that is specified as the "External network" in the base partition in cmsh (the output of "cmsh -c "partition use base; get externalnetwork"").

Modifying the value of the ResolveToExternalName directive and restarting the CMDaemon while important system services (e.g. Slurm) are running should not be done. Doing so is likely to cause problems with accessing such services due to them then running with a different domain name than the one with which they originally started.

On a tangential note that is closely, but not directly related to the ResolveToExternalName directive: the cluster can be configured so that the "hostname -f" command executed on a regular node returns the FQDN of that node, and so that the FQDN in turn resolves to an external IP for that regular node. The details on how to do this are in the Knowledge Articles of the NVIDIA Enterprise Support Portal at https://enterprise-support.nvidia.com/. A search query for FQDN leads to the relevant entry.

ResolveMasterToExternalName directive

Syntax: AdvancedConfig = {"ResolveMasterToExternalName=0|1", ...}
Default: ResolveMasterToExternalName = 1

ResolveMasterToExternalName is a parameter of the AdvancedConfig (page 925) directive.

ResolveMasterToExternalName can only be used if ResolveToExternalName (page 923) is active.

If set to 1 (the default), then the master head node, as specified by the name master, resolves to the IP address as set by ResolveToExternalName.

If set to 0 then the master head node resolves to the internal shared IP address.

ResolveMasterToExternalDomainName directive

Syntax: AdvancedConfig = {"ResolveMasterToExternalDomainName=0|1", ...}
Default: ResolveMasterToExternalDomainName = 1

ResolveMasterToExternalDomainName is a parameter of the AdvancedConfig (page 925) directive.

ResolveMasterToExternalDomainName can only be used if ResolveToExternalName (page 923) is active.

The external domain name as defined for the external network, in the form master.<*external domain*>, can be used in /etc/hosts during name resolution.

If set to 1 (the default), then the external domain name is used in /etc/hosts.

If set to 0, then the external domain name is not used in /etc/hosts.

Resolution occurs as shown in the following table:

ResolveMasterToExternalDomainName Directive Effects

on simple head, resolving:		on failover head, resolving:			on regular node, resolving:		Using the
master	head	master	head1	head2	master	head(s)	DNS?
ResolveToExternalName = False							
I	I	I	I	I	I	I	No
I	I	I	I	I	I	I	Yes
ResolveToExternalName = True							
E	E	I	E	E	I	I	No
E	E	I	E	E	I	E	Yes

Key: I: resolves to internal IP address of head

E: resolves to external IP address of head

DisableLua directive

Syntax: DisableLua = true|false
Default: DisableLua = false

The value of the DisableLua directive determines if Lua code (section L.6) used in monitoring expressions can be executed.

AdvancedConfig directive

Syntax: AdvancedConfig = { "<key1>=<value1>", "<key2>=<value2>", ... }
Default: *Commented out in the default* cmd.conf *file*

The AdvancedConfig directive is not part of the standard directives. It takes a set of key/value pairs as parameters, with each key/value pair allowing a particular functionality, and is quite normal in that respect. However, the functionality of a parameter to this directive is often applicable only under restricted conditions, or for non-standard configurations. The AdvancedConfig parameters are therefore generally not recommended for use by the administrator, nor are they generally documented.

Like for the other directives, only one AdvancedConfig directive line is used. This means that whatever functionality is to be enabled by this directive, its corresponding parameters must be added to that one line. These key/value pairs are therefore added by appending them to any existing AdvancedConfig key/value pairs, which means that the directive line can be a long list of key/value pairs to do with a variety of configurations.

Managing Key/Value Pairs With The cm-manipulate-advanced-config.py Utility

The cm-manipulate-advanced-config.py utility can be used to make it easier to manage AdvancedConfig key/value pairs.

For example, to add a key/value pair key8=value8:

```
[root@basecm11 ~]# cm-manipulate-advanced-config.py key8=value8
Updated: /cm/local/apps/cmd/etc/cmd.conf
```

To show the current state of the AdvancedConfig, the -s|--show option can be used:

```
[root@basecm11 ~]# cm-manipulate-advanced-config.py -s
=== /cm/local/apps/cmd/etc/cmd.conf ===
VirtualCluster=1
key8=value8
```

A key/value pair can be removed by specifying its key with the -r|--remove option:

```
[root@basecm11 ~]# cm-manipulate-advanced-config.py -r key8
Updated: /cm/local/apps/cmd/etc/cmd.conf
[root@basecm11 ~]# cm-manipulate-advanced-config.py -s
=== /cm/local/apps/cmd/etc/cmd.conf ===
VirtualCluster=1
```

The utility can be used on cmd.conf in node images too, using the -i|--image option.

```
[root@basecm11 ~]# cm-manipulate-advanced-config.py -i /cm/images/default-image
Updated: /cm/images/default-image/cm/local/apps/cmd/etc/cmd.conf
```

The -q option causes the utility to exit with code 1 if cmd.conf has changed.

Further options can be seen with the -h|--help option.

GlobalConfig directive

Syntax: GlobalConfig = { "<key1>=<value1>", "<key2>=<value2>", ... }
Default: *not in the default* cmd.conf *file*

The GlobalConfig directive is not part of the standard directives. It takes a set of key/value pairs as parameters, with each key/value pair allowing a particular functionality, and is quite normal in that respect. However, the parameter to this directive only needs to be specified on the head node. The non-head node CMDaemons take this value upon connection, which means that the cmd.conf file on the non-head nodes do not need to have this specified.

This allows nodes to set up, for example, their search domains using the MaximalSearchDomains GlobalConfig directive (page 917).

Like for the other directives, only one GlobalConfig directive line is used. This means that whatever functionality is to be enabled by this directive, its corresponding parameters must be added to that one line. These key/value pairs are therefore added by appending them to any existing GlobalConfig key/value pairs, which means that the directive line can be a long list of key/value pairs to do with a variety of configurations.

ScriptEnvironment directive

Syntax: ScriptEnvironment = { "CMD_ENV1=<value1>", "CMD_ENV2=<value2>", ... }
Default: *Commented out in the default* cmd.conf *file*

The ScriptEnvironment directive sets extra environment variables for CMDaemon and child processes.

For example, if CMDaemon is running behind a web proxy, then the environment variable http_proxy may need to be set for it. If, for example, the proxy is the host brawndo, and it is accessed via port 8080 using a username/password pair of joe/electrolytes, then the directive becomes:

```
ScriptEnvironment = { "http_proxy=joe:electrolytes@brawndo:8080" }
```

BurnSpoolDir directive

Syntax: `BurnSpoolDir = path`
Default: `BurnSpoolDir = "/var/spool/burn/"`

The `BurnSpoolDir` directive specifies the directory under which node burn log files are placed (Chapter 11 of the *Installation Manual*). The log files are logged under a directory named after the booting MAC address of the NIC of the node. For example, for a MAC address of 00:0c:29:92:55:5e the directory is `/var/spool/burn/00-0c-29-92-55-5e`.

IdleThreshold directive

Syntax: `IdleThreshold = number`
Default: `IdleThreshold = 1.0`

The `IdleThreshold` directive sets a threshold value for `loadone`. If `loadone` exceeds this value, then data producers that have `Only when idle` (page 588) set to `true` (`enabled`), will not run. If the data producer is sampled on a regular node rather than on the head node, then `cmd.conf` on the regular node should be modified and its CMDaemon restarted.

MonitoringPath directive

Syntax: `AdvancedConfig = {"MonitoringPath=path", ...}`
Default: Implicit value: `"MonitoringPath=/var/spool/cmd/monitoring/"`

`MonitoringPath` is a parameter of the `AdvancedConfig` (page 925) directive.

Its value determines the path of the directory in which monitoring data is saved (section 14.8).

MonitoringEntityActionHistoryDisabled directive

Syntax: `AdvancedConfig = {"MonitoringEntityActionHistoryDisabled =0|1"}`
Default: `0`

`MonitoringEntityActionHistoryDisabled` is a parameter of the `AdvancedConfig` (page 925) directive.

If it is set to a value of `1` then the action history (section 10.6.6) is disabled.

MaxServiceFailureCount directive

Syntax: `AdvancedConfig = {"MaxServiceFailureCount=number", ...}`
Default: Implicit value: `"MaxServiceFailureCount=10"`

`MaxServiceFailureCount` is a parameter of the `AdvancedConfig` (page 925) directive.

Its value determines the number of times a service failure event is logged (page 187). Restart attempts on the service still continue when this number is exceeded.

InitdScriptTimeout directive

Syntax: `AdvancedConfig = {"InitdScriptTimeout[.service]=timeout", ...}`
Default: Implicit value: `"InitdScriptTimeout=30"`

`InitdScriptTimeout` is a parameter of the `AdvancedConfig` (page 925) directive. It can be set globally or locally:

- **Global (all services)**
  `InitdScriptTimeout` can be set as a global value for init scripts, by assigning *timeout* as a period in seconds. If an init script fails to start up its service within this period, then CMDaemon kills the service and attempts to restart it.
  - If `InitdScriptTimeout` has a value for *timeout* set, then all init scripts have a default timeout of *timeout* seconds.
  - If `InitdScriptTimeout` has no *timeout* value set, then all init scripts have a default timeout of 30 seconds.
- **Local (for a specific service)**
  If `InitdScriptTimeout.`*service* is assigned a *timeout* value, then the init script for that *service* times out in *timeout* seconds. This timeout overrides, for that service only, any existing global default timeout.

When a timeout happens for an init script attempting to start a service, the event is logged. If the number of restart attempts exceeds the value determined by the `MaxServiceFailureCount` directive (page 927), then the event is no longer logged, but the restart attempts continue.

Example

An fhgfs startup takes a bit longer than 30 seconds, and therefore times out with the default timeout value of 30s. This results in the following logs in `/var/log/cmdaemon`:

```
cmd: [SERVICE] Debug: ProgramRunner: /etc/init.d/fhgfs-client restart
[DONE] 0 9
cmd: [SERVICE] Debug: /etc/init.d/fhgfs-client restart, exitcode = 0,
signal = 9
```

Here, *service* is `fhgfs-client`, so setting the parameter can be done with:

```
AdvancedConfig = { ..., "initdScriptTimeout.fhgfs-client=60", ...}
```

This allows a more generous timeout of 60 seconds instead.

Restarting CMDaemon then should allow the fhgs startup to complete

```
# systemctl restart cmd
```

A more refined approach that avoids a complete CMDaemon restart would be to execute a `reset` (page 187) on the `fhgfs-client` from within CMDaemon, as follows:

```
[basecm11->category[default]->services[fhgfs-client]]% reset fhgfs-client
Successfully reset service fhgfs-client on: node001,node002
[basecm11->category[default]->services[fhgfs-client]]%
```

CMDaemonListenOnInterfaces directive

Syntax: `AdvancedConfig = {"CMDaemonListenOnInterfaces=<`*interfaces*`>", ...}`
Default: *all interfaces listening to port 8081*

`CMDaemonListenOnInterfaces` is a parameter of the `AdvancedConfig` (page 925) directive.

When set explicitly, CMDaemon listens only to the interfaces listed in <*interfaces*>. The form of <*interfaces*> is a comma-separated list of interface device names:

Example

```
CMDaemonListenOnInterfaces=eth0,eth1,eth0:0,eth0:1
```

If the interface list item `lo` is omitted from the list of names, it will still be listened to. This is because CMDaemon must always be able to talk to itself on the loopback interface.

DisableInotifyInterface directive

Syntax: `AdvancedConfig = {"DisableInotifyInterface =0|1"}`
Default: `0`

`DisableInotifyInterface` is a parameter of the `AdvancedConfig` (page 925) directive.

When set to `1` CMDaemon ignores inotify events from all interfaces.

When set to `0` CMDaemon does not ignore inotify events from all interfaces.

IgnoreInotifyInterface directive

Syntax: `AdvancedConfig = {"IgnoreInotifyInterface =<`*interfaces*`>", ...}`
Syntax: `GlobalConfig = {"IgnoreInotifyInterface =<`*interfaces*`>", ...}`
Default: `veth*,lxc*,fg-*,qg-*,qr-*,sg-*,tbr-*,qbr*,qvb*,qvo*,cali*,flannel*,ecu_*,chassis_*, enx*,az*,tap*,rdma*,cni0,/[a-f0-9]12_[hc]/`

In the preceding lines showing the value for `Default`*, the line ending with* `chassis_*`*, must be concatenated to the line starting with* `enx*`*,*

`IgnoreInotifyInterface` is a parameter of the `AdvancedConfig` (page 925) directive, as well as the `GlobalConfig` (page 926) directive.

`IgnoreInotifyInterface` is active if `DisableInotifyInterface` is `0`.

By default, the directive has CMDaemon ignore the normally harmless messages that are typically generated by Kubernetes as Calico creates and removes interfaces.

CMDaemon ignores events from the interfaces listed in <*interfaces*>. The form of <*interfaces*> is a comma-separated list of interface device names, with some globbing allowed.

CookieCooldownTime directive

Syntax: `AdvancedConfig = {"CookieCooldownTime=`*number from 60 to 86400*`", ...}`
Default: `900`

`CookieCooldownTime` is a parameter of the `AdvancedConfig` (page 925) directive.

It defines the number of seconds until the Base View connection to CMDaemon times out, if there is no user activity at the Base View client side.

DHCPMaxleaseTime directive

Syntax: `AdvancedConfig = {"DHCPMaxleaseTime=`*number*`", ...}`
Default: *client default*

`DHCPMaxleaseTime` is a parameter of the `AdvancedConfig` (page 925) directive.

`DHCPMaxleaseTime` sets `max-lease-time` in DHCPOFFER. This is the maximum lease time, in seconds, that the DHCP server on the head node allows to the DHCP client on the node.

SlurmStraightExtraTopology directive

Syntax: `AdvancedConfig = {"SlurmStraightExtraTopology=0|1", ...}`
Default: `1`

`SlurmStraightExtraTopology` is a parameter of the `AdvancedConfig` (page 925) directive. If set to `0`, then the order of switches is reversed when setting extra values for `SlurmTopology` (page 412).

SlurmConcatTopologySwitchName directive

Syntax: `AdvancedConfig = {"SlurmConcatTopologySwitchName=0|1", ...}`
Default: `0`

`SlurmConcatTopologySwitchName` is a parameter of the `AdvancedConfig` (page 925) directive. If set to `1`, then it allows concatenation of the switch names in the topology defined via the `SlurmTopology` extra values (page 412).

JobsSamplingMetricsInterval directive

Syntax: `AdvancedConfig = {"JobsSamplingMetricsInterval=<`*number*`>", ...}`
Default: `60`

`JobsSamplingMetricsInterval` is a parameter of the `AdvancedConfig` (page 925) directive. Its value is a time period, in seconds, and it applies only to metrics associated with job queues. Such metric sampling is carried out with this time period if job queues are added, or if job queues are re-created after disappearing.

MembershipQueryInterval directive

Syntax: `AdvancedConfig = {"MembershipQueryInterval=<`*number*`>", ...}`
Default: `4`
`MembershipQueryInterval` is a parameter of the `AdvancedConfig` (page 925) directive. Its value is a time period, in seconds. This time period value elapses between the checks that CMDaemon makes to determine the node states (section 5.5) in a cluster. If the network is very congested, then a larger value can be used to reduce the network load caused by these checks.

AddUserPasswordReset directive

Syntax: `AdvancedConfig = {"AddUserPasswordReset=0|1", ...}`
Default: `0`

`AddUserPasswordReset` is a parameter of the `AdvancedConfig` (page 925) directive. If this parameter is set to `1`, then an LDAP user added after the change is in effect is forced to change their password on first login:

Example

```
root@basecm11:~# cm-manipulate-advanced-config.py -g AddUserPasswordReset=1
Updated: /cm/local/apps/cmd/etc/cmd.conf
root@basecm11:~# service cmd restart

root@basecm11:~# cmsh
[basecm11]% user
[basecm11->user]% add joe
[basecm11->user*[joe*]]% set password joe
[basecm11->user*[joe*]]% commit
[basecm11->user[joe]]%

root@basecm11:~# ssh joe@localhost
joe@localhost's password:
Password must be changed
...
WARNING: Your password has expired.
```

```
You must change your password now and login again!
(current) LDAP Password:
New password:
Retype new password:
passwd: password updated successfully
Connection to localhost closed.

root@basecm11:~# ssh joe@localhost
joe@localhost's password:
joe@basecm11:~$
```

With the AdvancedConfig configuration in effect, an already-existing user can also be forced to change password on the next login, after the extra variable pwdReset is set to true for that already-existing user. For example, for a user fred, executing the following:

Example

```
[basecm11->user[fred]]% set -e pwdReset true
[basecm11->user*[fred*]]% commit
```

means that fred is forced to change password on the next login.

AddUserScript directive

Syntax: AdvancedConfig = {"AddUserScript=<*path*>", ...}
Default: *none*

AddUserScript is a parameter of the AdvancedConfig (page 925) directive. If this parameter is set to a path leading to a script, and if a new user is added using cmsh or Base View, then the script is automatically run by CMDaemon, with the username of the new user automatically passed as the first argument to the script. The script has a default timeout of 5 seconds.

AddUserScriptPasswordInEnvironment directive

Syntax: AdvancedConfig = {"AddUserScriptPasswordInEnvironment=0|1", ...}
Default: 0

AddUserScriptPasswordInEnvironment is a parameter of the AdvancedConfig (page 925) directive. If this parameter is set to 1, then CMDaemon passes the CMD_USER_PASSWORD environment variable to the script defined by the AddUserScript directive.

RemoveUserScript directive

Syntax: AdvancedConfig = {"RemoveUserScript=<*path*>", ...}
Default: *none*

RemoveUserScript is a parameter of the AdvancedConfig (page 925) directive. If this parameter is set to a path leading to a script, and if an existing user is removed using cmsh or Base View, then the script is automatically run by CMDaemon. The script has a default timeout of 5 seconds.

AddUserScriptTimeout directive

Syntax: AdvancedConfig = {"AddUserScriptTimeout=<*number*>", ...}
Default: 5

AddUserScriptTimeout is a parameter of the AdvancedConfig (page 925) directive. It sets the timeout value in seconds, for the script set by AddUserScript.

RemoveUserScriptTimeout directive

Syntax: AdvancedConfig = {"RemoveUserScriptTimeout=<*number*>", ...}
Default: 5

RemoveUserScriptTimeout is a parameter of the AdvancedConfig (page 925) directive. It sets the timeout value in seconds, for the script set by RemoveUserScript.

AutomaticMountAll directive

Syntax: AdvancedConfig = {"AutomaticMountAll=0|1", ...}
Default: 1

The AutomaticMountAll directive is a parameter of the AdvancedConfig (page 925) directive. If set to the default of 1, then a mount -a operation is carried out when a mount change is carried out by CMDaemon.

The mount -a operation has to do with attempting to mount devices listed in /etc/fstab. It should not be confused with auto-mounting of filesystems, which has to do with mounting an arbitrary device to a filesystem automatically.

If the AutomaticMountAll directive is set to 0, then /etc/fstab is written, but the mount -a command is not run by CMDaemon. However, the administrator should be aware that since mount -a is run by the distribution during booting, a node reboot implements the mount change.

The directive is only set in the cmd.conf file of the compute nodes, and not the head node(s).

AllowImageUpdateWithAutoMount directive

Syntax: AdvancedConfig = {"AllowImageUpdateWithAutoMount=0|1|2|3", ...}
Default: 0

The AllowImageUpdateWithAutoMount directive is a parameter of the AdvancedConfig (page 925) directive. The values it takes decide how an auto-mounted filesystem should be dealt with during image updates (section 5.6.2) or grab to image (syncing node-to-image, section 9.5.2). It must be set in cmd.conf per software image or per node where the automount is running, and the changes activated as usual, as described on page 905.

Value	Description
0	If auto-mount is running, abort provisioning (default)
1	If auto-mount is running, warn but continue
2	Do not check auto-mount status. This saves a little time, but it risks data loss, unless the automounted filesystem has been added to excludelistupdate. If the automounted filesystem is not added to excludelistupdate, then if the automounted filesystem happens to be unavailable at the time that an image update is carried out, then the rsync process can end up deleting the automounted filesystem contents during the rsync, because it assumes that content should not be there.
3	Pretend auto-mount is running. This prevents an image update

How an auto-mounted filesystem can be configured using the autofs service in BCM is discussed

in section 3.14. The need for adding the automounted filesystem to excludelistupdate is discussed on page 298.

DNS::options_allow-query directive

Syntax: AdvancedConfig = {"DNS::options_allow-query=<*subnet1*>,<*subnet2*>,...",...}
Default: *unset*

The DNS::options_allow-query directive is a parameter of the AdvancedConfig (page 925) directive. If a subnet value is specified, in CIDR notation, then that subnet can query the DNS running on the head node. Setting a subnet places an entry within the allow-query section of /etc/named.conf. Extra subnets must specified with comma-separation, without whitespace.

The standard directive PublicDNS (page 917) simply adds the entry 0.0.0.0/0 to the allow-query section and can be used if no specific subnet needs to be added.

CipherList directive

Syntax: AdvancedConfig = {"CipherList=<*ciphers*>", ...}
Default: CipherList=ALL:!aNULL:!eNULL

The CipherList directive is a parameter of the AdvancedConfig (page 925) directive. It sets the cipher list of the OpenSSL suite that CMDaemon negotiates with clients. Ciphers in the cipher list can be viewed with:

Example

```
[root@basecm11 ~]# openssl ciphers -v
ECDHE-RSA-AES256-GCM-SHA384 TLSv1.2 Kx=ECDH     Au=RSA  Enc=AESGCM(256) Mac=AEAD
ECDHE-ECDSA-AES256-GCM-SHA384 TLSv1.2 Kx=ECDH     Au=ECDSA Enc=AESGCM(256) Mac=AEAD
ECDHE-RSA-AES256-SHA384 TLSv1.2 Kx=ECDH     Au=RSA  Enc=AES(256)  Mac=SHA384
ECDHE-ECDSA-AES256-SHA384 TLSv1.2 Kx=ECDH     Au=ECDSA Enc=AES(256)  Mac=SHA384
ECDHE-RSA-AES256-SHA    SSLv3 Kx=ECDH     Au=RSA  Enc=AES(256)  Mac=SHA1
...
```

The columns are: the cipher name, SSL protocol, key exchange used, authentication mechanism, encryption used, and MAC digest algorithm.

Ciphers can be specified for the CipherList directive according to the specification described in man ciphers.1. For example, as:

Example

```
AdvancedConfig = {"CipherList=ALL:!aNULL:!ADH:!eNULL:!LOW:!EXP:RC4+RSA:+HIGH:+MEDIUM"}
```

An analogous non-CMDaemon directive is found in LDAP client-server negotiation, where TLSCipherSuite can be set in the slapd.conf file, described in man slapd.conf.5.

SSLServerMethod directive (with TLS masking)

Syntax: AdvancedConfig = {"SSLServerMethod=TLS <*versionnumber*>", ...}
Syntax: GlobalConfig = {"SSLServerMethod=TLS <*versionnumber*>", ...}
Masking syntax: AdvancedConfig = {"TLS <*versionnumber*>=<*boolean*>", ...}
Masking syntax: GlobalConfig = {"TLS <*versionnumber*>=<*boolean*>", ...}
Default: GlobalConfig = { "TLS 1.2=1", "TLS 1.3=1", ... }

The SSLServerMethod directive is a parameter of the AdvancedConfig (page 925) directive, as well as the GlobalConfig (page 926) directive. It sets the SSL server method for the OpenSSL suite that CMDaemon negotiates with clients. Possible values for <*versionnumber*> are:

- 1
- 1.1
- 1.2
- 1.3

By default, with no SSLServerMethod directive, TLS versions 1.2 and 1.3 are enabled.

This is because there are still some SSL clients that require a TLS 1.2 cipher and have no TLS 1.3 cipher negotiation ability.

If the SSLServerMethod directive is specified, then only that TLS version is negotiated. For example, to allow only TLS 1.2 negotiation:

Example

```
GlobalConfig = { "SSLServerMethod=TLS 1.2", ... }
```

The SSLServerMethod directive is implied by the TLS masking syntax. The TLS masking syntax allows TLS negotiation methods to be set concurrently. For example, TLS 1 and TLS1.1 could be disabled, and TLS 1.2 and TLS 1.3 could be enabled, with:

Example

```
GlobalConfig = { "TLS 1=0", "TLS 1.1=0", "TLS 1.2=1", "TLS 1.3=1", ... }
```

The GlobalConfig value is the directive that the cluster administrator should set in most cases. It only needs to be set on cmd.conf on the head node (or head nodes in a high availability configuration), in order to configure the same TLS negotiation settings for the regular nodes.

The GlobalConfig value is overridden by the AdvancedConfig value. The AdvancedConfig value needs to be set on cmd.conf on the head node(s), but also needs to be set on the non-head nodes that are to be disabled. Setting the AdvancedConfig directive allows custom configuration of the TLS negotiations to be carried out per node, and if the negotiations do not match with the other end, then other settings are tried. This does mean that sometimes nodes with the extra settings end up running extra processes for no reason. Running the directive as an AdvancedConfig is therefore suboptimal for most reasonable use cases.

Reverting to the TLS 1.1 fallback cipher availability of earlier versions of CMDaemon is possible by setting:

```
AdvancedConfig = { "TLS 1.1=1", "TLS 1.2=1", "TLS 1.3=1" }
```

Setting this brings with it the risk of allowing cryptographic downgrade attacks. It is therefore not recommended, and is also why it is disabled by default.

JobInformationDisabled directive

Syntax: AdvancedConfig = {"JobInformationDisabled=0|1", ...}
Syntax: GlobalConfig = {"JobInformationDisabled=0|1", ...}
Default: JobInformationDisabled=0

The JobInformationDisabled directive is a parameter of the AdvancedConfig (page 925) directive, as well as the GlobalConfig (page 926) directive. If set to 1 it disables job-based monitoring (Chapter 11). The details of how this is done are discussed shortly.

For a cluster that is running millions of jobs at a time, job-based monitoring can typically consume significant resources. The monitoring of data for so many small jobs is typically not useful and not of interest. For such a case, setting this directive improves cluster performance by not having to deal with the information on the current jobs running on the cluster.

The `GlobalConfig` value is the directive that the cluster administrator should set in most cases. It only needs to be set on `cmd.conf` on the head node (or head nodes in a high availability configuration). Setting it disables the collection of job information by the nodes running the jobs, and no collection of the job information from those nodes is done by the head (monitoring) node.

The `GlobalConfig` value is overridden by the `AdvancedConfig` value. The `AdvancedConfig` value needs to be set on `cmd.conf` on the head node(s), but also needs to be set on the non-head nodes that are to be disabled. Setting the `AdvancedConfig` directive still allows job information to be collected by the nodes running the jobs, but the information is not collected from the nodes by the head (monitoring) node. This means that nodes are running extra processes for probably no reason. Running the directive as an `AdvancedConfig` is therefore suboptimal for most use cases.

JobInformationKeepDuration directive

Syntax: `AdvancedConfig = {"JobInformationKeepDuration=<`*number*`>", ...}`
Default: `JobInformationKeepDuration=2419200`

The `JobInformationKeepDuration` directive is a parameter of the `AdvancedConfig` (page 925) directive. It takes on a value in seconds. If a job has finished more than that many seconds ago, then it will be removed along with all its monitoring data from the database. By default it is set to 28 days ($24 \times 3600 \times 28$ seconds).

If `persistent` is set to `yes` (`cmsh:wlm > jobs`), then job information is not removed even after `JobInformationKeepDuration` has been exceeded.

JobInformationChargeBackKeepDuration directive

Syntax: `AdvancedConfig = {"JobInformationChargeBackKeepDuration=<`*number*`>", ...}`
Default: `JobInformationChargeBackKeepDuration=158112000`

The `JobInformationChargeBackKeepDuration` directive is a parameter of the `AdvancedConfig` (page 925) directive. It takes on a value in seconds. If a job has finished more than that many seconds ago, then it will be removed along with all its monitoring data from the database. By default it is set to a little more than 5 years ($24 \times 3600 \times 366 \times 5$ seconds).

JobInformationKeepCount directive

Syntax: `AdvancedConfig = {"JobInformationKeepCount=<`*number*`>", ...}`
Default: `JobInformationKeepCount=8192`

The `JobInformationKeepCount` directive is a parameter of the `AdvancedConfig` (page 925) directive. If the total number of jobs is greater than `(JobInformationKeepCount + 10%)`, then the job record and its data content are discarded, starting from the oldest job first, until the total number of jobs remaining becomes `JobInformationKeepCount`. If it is set to 0, then none of the job records and content are removed.

The maximum value for this directive is `65536`. A larger value is treated as `65536`.

JobInformationChargeBackKeepCount directive

Syntax: `AdvancedConfig = {"JobInformationChargeBackKeepCount=<`*number*`>", ...}`

Default: JobInformationChargeBackKeepCount=1048576

The JobInformationChargeBackKeepCount directive is a parameter of the AdvancedConfig (page 925) directive. If the total number of jobs is greater than (JobInformationChargeBackKeepCount + 10%), then the job record and its data content are discarded, starting from the oldest job first, until the total number of jobs remaining becomes JobInformationChargeBackKeepCount. If it is set to 0, then none of the job records and content are removed.

The default value of about 1 million corresponds to about 0.5 GB of storage.

The maximum value for this directive is about 1 billion (1,073,741,824), which corresponds to about 500 GB of storage.

JobInformationMinimalJobDuration directive

Syntax: AdvancedConfig = {"JobInformationMinimalJobDuration=<*number*>", ...}
Default: JobInformationMinimalJobDuration=0

The JobInformationMinimalJobDuration directive is a parameter of the AdvancedConfig (page 925) directive. If set, then jobs that run for less than this number of seconds are not stored in the cache. Its default value of 0 seconds means that all jobs are handled.

JobInformationFlushInterval directive

Syntax: AdvancedConfig = {"JobInformationFlushInterval=<*number*>", ...}
Default: JobInformationFlushInterval=600

The JobInformationFlushInterval directive is a parameter of the AdvancedConfig (page 925) directive. If this interval, in seconds, is set, then the cache is flushed to the database with that interval. Its default value is 10 minutes (10×60 seconds). Values of around 30 seconds or less will conflict with the default CMDaemon maintenance timeout value of 30 seconds, and will mostly simply add load.

JobInformationChargeBackRemoveInterval directive

Syntax: AdvancedConfig = {"JobInformationChargeBackRemoveInterval=<*number*>", ...}
Default: JobInformationChargeBackRemoveInterval=600

The JobInformationChargeBackRemoveInterval directive is a parameter of the AdvancedConfig (page 925) directive. If this interval, in seconds, is set, then the cache is flushed to the database with that interval. Its default value is 10 minutes (10×60 seconds). Values of around 30 seconds or less will conflict with the default CMDaemon maintenance timeout value of 30 seconds, and will mostly simply add load.

The maximum remove interval is 1 day (86400 seconds).

ActionLoggerFile directive

Syntax: AdvancedConfig = {"ActionLoggerFile=*filename*", ...}
Default: /var/spool/cmd/actions.log

The directive is a parameter of the AdvancedConfig (page 925) directive. Its value overrides the default path.

The directive needs to be implemented per node or image.

ActionLoggerOnSuccess directive

Syntax: AdvancedConfig = {"ActionLoggerOnSuccess=0|1", ...}

Default: `ActionLoggerOnSuccess=0`

The directive is a parameter of the `AdvancedConfig` (page 925) directive.
By default, only failed actions are logged. Successful actions are also logged if setting `ActionLoggerOnSuccess=1`

The directive needs to be implemented per node or image.

The log file shows timestamped output with one line per run for an action script, with the script response.

Example

```
(time)  /path/to/action/script [ timeout ]
(time)  /path/to/action/script [ failed ] (exit code: 1)
(time)  /path/to/action/script [ success ]
```

The `success` line only appears if `ActionLoggerOnSuccess=1`.

FailoverPowerRetries directive

Syntax: `AdvancedConfig = {"FailoverPowerRetries=<`*number*`>", ...}`
Default: `FailoverPowerRetries=5`

The `FailoverPowerRetries` directive is a parameter of the `AdvancedConfig` (page 925) directive.

After a decision to carry out the failover has been made, CMDaemon sends a power off command to the BMC of the head node that is meant to be powered off. If the power off command fails, then on getting the fail response, CMDaemon waits for 1 second. After that second, it sends out a power off command again. The value of `FailoverPowerRetries` is the number of times that CMDaemon retries sending the power off command to the BMC of the head node that is intended to be powered off during the failover, if the response from the power off command remains a fail response.

The power down attempts cease, either when the BMC reports that the head node is OFF, or when the number of attempts reaches the `FailoverPowerRetries` value.

`FailoverPowerRetries` takes a maximum value of 120.

A value of 0 means that 1 attempt to power off is carried out during failover, but no retry is attempted after the first attempt.

Because CMDaemon waits for a period of 1s before checking for an OFF reponse, it means that the number of retries is about the same as the number of seconds before CMDaemon decides that powering off has failed, unless the OFF response also takes some time to get to CMDaemon.

Increasing the value for this directive can be useful for some BMC cards that take longer than about 5s to report their power status, because the power off attempt may otherwise time out.

AddUserDefaultGroupID directive

Syntax: `AdvancedConfig = { "AddUserDefaultGroupID = <`*number*`>", ...}`
Default: *none*

The `AddUserDefaultGroupID` directive is a parameter of the `AdvancedConfig` (page 925) directive.

If the `AddUserDefaultGroupID` is unset, then the default group ID of a new user is the same as the User ID.

If the `AddUserDefaultGroupID` is set, then the set value becomes the default GUID of new users, when new users are created via the `cmsh` or Base View front ends to CMDaemon.

The directive is not intended to set a non-default group occasionally during user creation. In that case, a non-default group ID can be set from the `cmsh` or Base View front ends, by setting the `groupid` value for the user.

To set a default group for a directory, while retaining the default group for the user, it may be possible to use the setgid bit for a directory.

LowUserId directive

Syntax: `AdvancedConfig = {"LowUserId=<number>", ...}`
Default: `1000` (`500` on SLES)

The `LowUserId` directive is a parameter of the `AdvancedConfig` (page 925) directive. From BCM 11.34 onwards, it controls the starting point for auto-allocated UIDs and GIDs. The default is `1000` (`500` on SLES), so behavior is unchanged for clusters that have not set this parameter explicitly.

Administrators who need UIDs and GIDs to start at a different value can set `LowUserId` in `cmd.conf`:

Example

```
AdvancedConfig = {"LowUserId=500"}
```

MaxMeasurablesPerProducer directive

Syntax: `AdvancedConfig = { "MaxMeasurablesPerProducer = <number>", ...}`
Default: `500`

The `MaxMeasurablesPerProducer` directive is a parameter of the `AdvancedConfig` (page 925) directive.

By default there is a software limit of 500 measurables per data producer. If this limit is exceeded, then the CMDaemon monitoring info logs show complaints about "too many measurables".

In that case, if the cluster hardware is not too slow, and if the measurables produced are not some kind of hardware garbage data values, then increasing the value of this directive should allow more measurables to be dealt with.

HeadNodeCertificateNameAlternatives directive

Syntax: `AdvancedConfig = {"HeadNodeCertificateNameAlternatives=<name,...>",...}`
Default: *none*

The `HeadNodeCertificateNameAlternatives` directive is a key of the `AdvancedConfig` (page 925) directive. It takes on a comma-separated list of alternative DNS names as values to add to the SSL certificate that it generates for CMDaemon. Extra names must specified with comma-separation, without white-space.

The old certificates must be removed before the new ones are used.

Example

```
[root@basecm11 ~]# cm-manipulate-advanced-config.py HeadNodeCertificateNameAlternatives=myname1,myname2
[root@basecm11 ~]# openssl x509 -in /cm/local/apps/cmd/etc/cert.pem -noout -text | grep -o DNS:myname1.*$
[root@basecm11 ~]# rm /cm/local/apps/cmd/etc/cert.pem,key
[root@basecm11 ~]# systemctl restart cmd    #new certificates auto-generated
[root@basecm11 ~]# openssl x509 -in /cm/local/apps/cmd/etc/cert.pem -noout -text | grep -o DNS:myname1.*$
DNS:myname1, DNS:myname1:8081, DNS:myname2, DNS:myname2:8081
```

D

Disk Partitioning And RAID Configuration

Disk partitioning is initially configured on the head node and regular nodes during installation (section 3.3.16 of the *Installation Manual*).

For the head node it cannot be changed from within BCM after implementation, and the head node is therefore not considered further in this section.

For regular nodes, partitioning can be changed after the initial configuration, by specifying a particular set of values according to the XML partitioning schema described in section D.1.

For example, for regular nodes, changing the value set for the XML tag of:

```
<xs:element name='filesystem'>
```

decides which filesystem type out of `ext2`, `ext3`, `ext4`, `xfs`, and so on, is used. The changes are implemented during the node partitioning stage of the node-installer (section 5.4.6).

Diskless operation can also be implemented by using an appropriate XML file. This is introduced in section 3.13.1.

Software or hardware RAID configuration can also be set for the regular nodes. The layouts must be specified following the XML schema files stored on the head node in the directory `/cm/node-installer/scripts/`:

- Software RAID configuration is set in the global partitioning XML schema file `disks.xsd` (section D.1).
- Hardware RAID configuration is set in the separate hardware RAID XML schema file `raid.xsd` (section D.2).

D.1 Structure Of Partitioning Definition—The Global Partitioning XML Schema Definition File

In BCM, regular node partitioning setups have their global structure specified using an XML schema definition file, which is installed on the head node in `/cm/local/apps/cmd/etc/htdocs/xsd/disks.xsd`.

The schema can also be viewed with:

Example

```
[root@basecm11 ]# cmsh -c "main xsdschema disks" | less
```

This schema does not include a hardware RAID definition. The hardware RAID schema is defined separately in the file `raid.xsd` (section D.2).

Examples of schemas in use, with and without hardware RAID, are given in sections D.3 and beyond.

An XML file can be validated against an XML schema definition file using the `xmllint` tool:

Example

```
[root@basecm11 ]# cd /cm/local/apps/cmd/etc/htdocs/xsd
[root@basecm11 xsd]# xmllint --noout --schema disks.xsd ../disk-setup/x86_64-slave-diskless.xml
../disk-setup/x86_64-slave-diskless.xml validates
[root@basecm11 xsd]#
```

XML schema for partitioning

```
<?xml version='1.0'?>

<!--
#
# SPDX-FileCopyrightText: Copyright (c) 2023 NVIDIA CORPORATION & AFFILIATES. All rights reserved.
# SPDX-License-Identifier: LicenseRef-NvidiaProprietary
#
# NVIDIA CORPORATION, its affiliates and licensors retain all intellectual
# property and proprietary rights in and to this material, related
# documentation and any modifications thereto. Any use, reproduction,
# disclosure or distribution of this material and related documentation
# without an express license agreement from NVIDIA CORPORATION or
# its affiliates is strictly prohibited.
#

This is the XML schema description of the partition layout XML file.
It can be used by software to validate partitioning XML files.
There are however a few things the schema does not check:
- There should be exactly one root mountpoint (/), unless diskless.
- There can only be one partition with a 'max' size on a particular device.
- Something similar applies to logical volumes.
- The 'auto' size can only be used for a swap partition.
- Partitions of type 'linux swap' should not have a filesystem.
- Partitions of type 'linux raid' should not have a filesystem.
- Partitions of type 'linux lvm' should not have a filesystem.
- Partitions of type 'unspecified' should not have a filesystem.
- If a raid is a member of another raid then it can not have a filesystem.
- Partitions, which are listed as raid members, should be of type 'linux raid'.
- If diskless is not set, there should be at least one device.
- The priority tag is only valid for partitions which have type set to
  "linux swap".
-->

<xs:schema xmlns:xs='http://www.w3.org/2001/XMLSchema' elementFormDefault='qualified'>

  <xs:element name='diskSetup'>

    <xs:complexType>
      <xs:sequence>
        <xs:element name='diskless' type='diskless' minOccurs='0' maxOccurs='1'/>
        <xs:element name='device' type='device' minOccurs='0' maxOccurs='unbounded'/>
        <xs:element name='raid' type='raid' minOccurs='0' maxOccurs='unbounded'/>
        <xs:element name='volumeGroup' type='volumeGroup' minOccurs='0' maxOccurs='unbounded'/>
        <xs:element name='subVolumes' type='subVolumes' minOccurs='0' maxOccurs='unbounded'/>
      </xs:sequence>
    </xs:complexType>
```

```
  <xs:key name='partitionAndRaidIds'>
    <xs:selector xpath='.//raid|.//partition'/>
    <xs:field xpath='@id'/>
  </xs:key>

  <xs:keyref name='raidMemberIds' refer='partitionAndRaidIds'>
    <xs:selector xpath='.//raid/member'/>
    <xs:field xpath='.'/>
  </xs:keyref>

  <xs:keyref name='volumeGroupPhysicalVolumes' refer='partitionAndRaidIds'>
    <xs:selector xpath='.//volumeGroup/physicalVolumes/member'/>
    <xs:field xpath='.'/>
  </xs:keyref>

  <xs:keyref name='subVolumeIds' refer='partitionAndRaidIds'>
    <xs:selector xpath='.//subVolumes'/>
    <xs:field xpath='@parent'/>
  </xs:keyref>

  <xs:unique name='raidAndVolumeMembersUnique'>
    <xs:selector xpath='.//member'/>
    <xs:field xpath='.'/>
  </xs:unique>

  <xs:unique name='deviceNodesUnique'>
    <xs:selector xpath='.//device/blockdev'/>
    <xs:field xpath='.'/>
    <xs:field xpath='@mode'/>
  </xs:unique>

  <xs:unique name='mountPointsUnique'>
    <xs:selector xpath='.//mountPoint'/>
    <xs:field xpath='.'/>
  </xs:unique>

  <xs:unique name='assertNamesUnique'>
    <xs:selector xpath='.//assert'/>
    <xs:field xpath='@name'/>
  </xs:unique>

</xs:element>

<xs:complexType name='diskless'>
  <xs:attribute name='maxMemSize' type='memSize' use='required'/>
  <xs:attribute name='mountOptions' type='xs:string'/>
</xs:complexType>

<xs:simpleType name='memSize'>
  <xs:restriction base='xs:string'>
    <xs:pattern value='([0-9]+[MG])|100%|[0-9][0-9]%|[0-9]%|0'/>
  </xs:restriction>
</xs:simpleType>
```

```
<xs:simpleType name='stripeSize'>
  <xs:restriction base='xs:string'>
    <xs:pattern value='4|8|16|32|64|128|256|512|1024|1K|2048|2K|4096|4K'/>
    <xs:pattern value='8192|8K|16384|16K|32768|32K|65536|64K|131072|128K'/>
    <xs:pattern value='262144|256K|524288|512K|1048576|1024K|1M'/>
    <xs:pattern value='2097152|2048K|2M|4194304|4096K|4M'/>
  </xs:restriction>
</xs:simpleType>

<xs:simpleType name='size'>
  <xs:restriction base='xs:string'>
    <xs:pattern value='max|auto|[0-9]+[MGT]'/>
  </xs:restriction>
</xs:simpleType>

<xs:simpleType name='relativeOrAbsoluteSize'>
  <xs:restriction base='xs:string'>
    <xs:pattern value='max|auto|[0-9]+[MGT]|[0-9]+([.][0-9]+)?%|[0-9]+/[0-9]+'/>
  </xs:restriction>
</xs:simpleType>

<xs:simpleType name='extentSize'>
  <xs:restriction base='xs:string'>
    <xs:pattern value='([0-9])+M'/>
  </xs:restriction>
</xs:simpleType>

<xs:simpleType name='blockdevName'>
  <xs:restriction base='xs:string'>
    <xs:pattern value='/dev/.+'/>
  </xs:restriction>
</xs:simpleType>

<xs:complexType name='blockdev'>
  <xs:simpleContent>
    <xs:extension base="blockdevName">
      <xs:attribute name="mode" default='normal'>
        <xs:simpleType>
          <xs:restriction base="xs:string">
            <xs:pattern value="normal|cloud|both"/>
          </xs:restriction>
        </xs:simpleType>
      </xs:attribute>
    </xs:extension>
  </xs:simpleContent>
</xs:complexType>

<xs:complexType name='device'>
  <xs:sequence>
    <xs:element name='blockdev' type='blockdev' minOccurs='1' maxOccurs='unbounded'/>
    <xs:element name='vendor' type='xs:string' minOccurs='0' maxOccurs='1'/>
    <xs:element name='requiredSize' type='size' minOccurs='0' maxOccurs='1'/>
    <xs:element name='assert' minOccurs='0' maxOccurs='unbounded'>
      <xs:complexType>
        <xs:simpleContent>
```

```
          <xs:extension base='xs:string'>
            <xs:attribute name='name' use='required'>
              <xs:simpleType>
                <xs:restriction base='xs:string'>
                  <xs:pattern value='[a-zA-Z0-9-_]+'/>
                </xs:restriction>
              </xs:simpleType>
            </xs:attribute>
            <xs:attribute name='args' type='xs:string'/>
          </xs:extension>
        </xs:simpleContent>
      </xs:complexType>
    </xs:element>
    <xs:element name='alignMiB' type='xs:boolean' minOccurs='0' maxOccurs='1'/>
    <xs:element name="partitionTable" minOccurs='0' maxOccurs='1'>
      <xs:simpleType>
        <xs:restriction base="xs:string">
          <xs:pattern value="gpt|msdos"/>
        </xs:restriction>
      </xs:simpleType>
    </xs:element>
    <xs:element name='partition' type='partition' minOccurs='1' maxOccurs='unbounded'/>
  </xs:sequence>
  <xs:attribute name='origin' type='xs:string'/>
</xs:complexType>

<xs:complexType name='partition'>
  <xs:sequence>
    <xs:element name='size' type='relativeOrAbsoluteSize'/>
    <xs:element name='type'>
      <xs:simpleType>
        <xs:restriction base='xs:string'>
          <xs:enumeration value='linux'/>
          <xs:enumeration value='linux swap'/>
          <xs:enumeration value='linux raid'/>
          <xs:enumeration value='linux lvm'/>
          <xs:enumeration value='unspecified'/>
        </xs:restriction>
      </xs:simpleType>
    </xs:element>
    <xs:element name='encryption' type='encryption' minOccurs='0' maxOccurs='1'/>
    <xs:group ref='filesystem' minOccurs='0' maxOccurs='1'/>
    <xs:element name='priority' minOccurs='0' maxOccurs='1'>
      <xs:simpleType>
        <xs:restriction base="xs:integer">
          <xs:minInclusive value="0"/>
          <xs:maxInclusive value="32767"/>
        </xs:restriction>
      </xs:simpleType>
    </xs:element>
  </xs:sequence>
  <xs:attribute name='id' type='xs:string' use='required'/>
  <xs:attribute name='partitiontype' type='xs:string'/>
</xs:complexType>
```

```
<xs:complexType name='encryption' mixed="true">
  <xs:sequence>
    <xs:element name='type' minOccurs='0' maxOccurs='1'>
      <xs:simpleType>
        <xs:restriction base='xs:string'>
          <xs:enumeration value='luks2'/>
          <xs:enumeration value='luks1'/>
          <!-- rest of this enum is intentionally left blank -->
        </xs:restriction>
      </xs:simpleType>
    </xs:element>
    <xs:element name='cipher' minOccurs='0' maxOccurs='1' type='xs:string'/>
    <xs:element name='hash' minOccurs='0' maxOccurs='1' type='xs:string'/>
    <xs:element name='keySize' minOccurs='0' maxOccurs='1' type='xs:integer'/>
    <xs:element name='name' minOccurs='0' maxOccurs='1' type='xs:string'/>
    <xs:element name='allowDiscards' minOccurs='0' maxOccurs='1' type='xs:boolean' default='true'/>
    <xs:element name='custom' minOccurs='0' maxOccurs='1' type='xs:string'/>
  </xs:sequence>
</xs:complexType>

<xs:group name='filesystem'>
  <xs:sequence>
    <xs:element name='filesystem'>
      <xs:simpleType>
        <xs:restriction base='xs:string'>
          <xs:enumeration value='ext2'/>
          <xs:enumeration value='ext3'/>
          <xs:enumeration value='ext4'/>
          <xs:enumeration value='xfs'/>
          <xs:enumeration value='btrfs'/>
          <xs:enumeration value='zfs'/>
          <xs:enumeration value='fat32'/>
          <xs:enumeration value='fat'/>
        </xs:restriction>
      </xs:simpleType>
    </xs:element>
    <xs:element name='mkfsFlags' type='xs:string' minOccurs='0' maxOccurs='1'/>
    <xs:element name='mountPoint' type='xs:string' minOccurs='0' maxOccurs='1'/>
    <xs:element name='mountOptions' type='xs:string' default='defaults' minOccurs='0'/>
  </xs:sequence>
</xs:group>

<xs:complexType name='raid'>
  <xs:sequence>
    <xs:element name='member' type='xs:string' minOccurs='2' maxOccurs='unbounded'/>
    <xs:element name='level' type='xs:int'/>
    <xs:element name='metaData' minOccurs='0' maxOccurs='1'>
      <xs:simpleType>
        <xs:restriction base='xs:string'>
          <xs:enumeration value='0'/>
          <xs:enumeration value='0.9'/>
          <xs:enumeration value='0.90'/>
          <xs:enumeration value='1'/>
          <xs:enumeration value='1.0'/>
          <xs:enumeration value='1.1'/>
```

```
          <xs:enumeration value='1.2'/>
          <xs:enumeration value='default'/>
        </xs:restriction>
      </xs:simpleType>
    </xs:element>
    <xs:element name='encryption' type='encryption' minOccurs='0' maxOccurs='1'/>
    <xs:choice minOccurs='0' maxOccurs='1'>
      <xs:group ref='filesystem'/>
      <xs:sequence>
        <xs:element name='swap'><xs:complexType /></xs:element>
        <xs:element name='priority' minOccurs='0' maxOccurs='1'>
          <xs:simpleType>
            <xs:restriction base="xs:integer">
              <xs:minInclusive value="0"/>
              <xs:maxInclusive value="32767"/>
            </xs:restriction>
          </xs:simpleType>
        </xs:element>
      </xs:sequence>
    </xs:choice>
  </xs:sequence>
  <xs:attribute name='id' type='xs:string' use='required'/>
</xs:complexType>

<xs:complexType name='volumeGroup'>
  <xs:sequence>
    <xs:element name='name' type='xs:string'/>
    <xs:element name='extentSize' type='extentSize'/>
    <xs:element name='physicalVolumes'>
      <xs:complexType>
        <xs:sequence>
          <xs:element name='member' type='xs:string' minOccurs='1' maxOccurs='unbounded'/>
        </xs:sequence>
      </xs:complexType>
    </xs:element>
    <xs:element name='logicalVolumes'>
      <xs:complexType>
        <xs:sequence>
          <xs:element name='volume' type='logicalVolume' minOccurs='1' maxOccurs='unbounded'/>
        </xs:sequence>
      </xs:complexType>
    </xs:element>
  </xs:sequence>
</xs:complexType>

<xs:complexType name='logicalVolume'>
  <xs:sequence>
    <xs:element name='name' type='xs:string'/>
    <xs:element name='size' type='size'/>
    <xs:element name='pool' type='xs:string' minOccurs='0' maxOccurs='1'/>
    <xs:element name='stripes' minOccurs='0' maxOccurs='1'>
      <xs:simpleType>
        <xs:restriction base="xs:integer">
          <xs:minInclusive value="1"/>
          <xs:maxInclusive value="32768"/>
```

```
          </xs:restriction>
        </xs:simpleType>
      </xs:element>
      <xs:element name='stripeSize' type='stripeSize' minOccurs='0' maxOccurs='1'/>
      <xs:element name='encryption' type='encryption' minOccurs='0' maxOccurs='1'/>
      <xs:element name='swap' minOccurs='0' maxOccurs='1'/>
      <xs:element name='priority' minOccurs='0' maxOccurs='1'>
        <xs:simpleType>
          <xs:restriction base="xs:integer">
            <xs:minInclusive value="0"/>
            <xs:maxInclusive value="32767"/>
          </xs:restriction>
        </xs:simpleType>
      </xs:element>
      <xs:group ref='filesystem' minOccurs='0' maxOccurs='1'/>
    </xs:sequence>
    <xs:attribute name="thinpool">
      <xs:simpleType>
        <xs:restriction base="xs:string">
          <xs:pattern value="1"/>
        </xs:restriction>
      </xs:simpleType>
    </xs:attribute>
    <xs:attribute name="metadatasize">
      <xs:simpleType>
        <xs:restriction base="xs:string">
          <xs:pattern value="([1-9][MGT])|([1-9][0-9]+[MGT])"/>
        </xs:restriction>
      </xs:simpleType>
    </xs:attribute>
  </xs:complexType>

  <xs:complexType name='subVolumes'>
    <xs:sequence>
      <xs:element name='subVolume' type='subVolume' minOccurs='1' maxOccurs='unbounded'/>
    </xs:sequence>
    <xs:attribute name='parent' type='xs:string' use='required'/>
  </xs:complexType>

  <xs:complexType name='subVolume'>
    <xs:sequence>
      <xs:element name='mountPoint' type='xs:string'/>
      <xs:element name='mountOptions' type='xs:string'/>
    </xs:sequence>
  </xs:complexType>

</xs:schema>
```

Examples Of Element Types In XML Schema

Name Of Type	Example Values
size	10G, 128M, 1T, 2.5T, 1/3, 33.333%, auto, max
device	/dev/sda, /dev/hda, /dev/cciss/c0d0
partition	linux, linux raid, linux swap, unspecified
filesystem	ext2, ext3, ext4, xfs

D.2 Structure Of Hardware RAID Definition—The Hardware RAID XML Schema Definition File

If a hardware RAID has already been created outside of BCM, then no XML definition is needed. This assumes that required kernel modules for the device load, so that the operating system ends up treating the RAID as a standard block device, which can therefore have its layout configured as described in section 3.13.

If, instead, hardware RAID is to be created and managed by BCM, then it can be specified using an XML schema definition file, stored on the head node in `/cm/local/apps/cmd/etc/htdocs/xsd/raid.xsd`.

The schema can also be viewed with `cmsh`:

Example

```
[root@basecm11 ]# cmsh -c "main xsdschema raid" | less
```

The full schema definition is listed next, while schema examples are listed in section D.4.1.

Configuration using BCM is currently limited to MegaRAID hardware. It may not work for newer controllers that do not support MegaCLI, or its successor StorCLI. It may also not work for controllers by vendors that do not use MegaCLI or StorCLI.

```
<?xml version="1.0" encoding="ISO-8859-1" ?>

<!--
#
# SPDX-FileCopyrightText: Copyright (c) 2023 NVIDIA CORPORATION & AFFILIATES. All rights reserved.
# SPDX-License-Identifier: LicenseRef-NvidiaProprietary
#
# NVIDIA CORPORATION, its affiliates and licensors retain all intellectual
# property and proprietary rights in and to this material, related
# documentation and any modifications thereto. Any use, reproduction,
# disclosure or distribution of this material and related documentation
# without an express license agreement from NVIDIA CORPORATION or
# its affiliates is strictly prohibited.
#

This is the XML schema description of the hardware RAID layout XML file.
It can be used by software to validate partitioning XML files.
There are however a few things the schema does not check:
- All of the spans (drive groups) in an raidArray must have the same number of drives.
- There can only be one volume with a 'max' size on a particular array, and this
must be the last volume in the array.
- If there is only one enclosure defined for a particular RAID controller the actual
enclosureID can be omitted, by using "auto" instead. Otherwise, the actual enclosureID
must be specified.
-->
```

```
<xs:schema xmlns:xs="http://www.w3.org/2001/XMLSchema">

<xs:simpleType name="raidLevel">
  <xs:restriction base="xs:nonNegativeInteger">
    <xs:pattern value="0|1|5|10|50"/>
  </xs:restriction>
</xs:simpleType>

<xs:simpleType name="volumeSize">
  <xs:restriction base="xs:string">
    <xs:pattern value="[0-9]{1,5}[MGT]|max"/>
  </xs:restriction>
</xs:simpleType>

<xs:simpleType name="stripeSize">
  <xs:restriction base="xs:string">
    <xs:pattern value="8K|16K|32K|64K|128K|256K|512K|1024K"/>
  </xs:restriction>
</xs:simpleType>

<xs:simpleType name="cachePolicy">
  <xs:restriction base="xs:string">
    <xs:pattern value="Cached|Direct"/>
  </xs:restriction>
</xs:simpleType>

<!--
  NORA  : No Read Ahead
  RA    : Read Ahead
  ADRA  : Adaptive Read
-->
<xs:simpleType name="readPolicy">
  <xs:restriction base="xs:string">
    <xs:pattern value="NORA|RA|ADRA"/>
  </xs:restriction>
</xs:simpleType>

<!--
  WT  : Write Through
  WB  : Write Back
-->
<xs:simpleType name="writePolicy">
  <xs:restriction base="xs:string">
    <xs:pattern value="WT|WB"/>
  </xs:restriction>
</xs:simpleType>

<xs:simpleType name="enclosureID">
  <xs:restriction base="xs:string">
    <xs:pattern value="auto|[0-9]{1,4}"/>
  </xs:restriction>
</xs:simpleType>

<xs:simpleType name="slotNumber">
  <xs:restriction base="xs:nonNegativeInteger">
```

```
    <xs:pattern value="[0-9]{1,2}"/>
  </xs:restriction>
</xs:simpleType>


<xs:element name="raidSetup">
  <xs:complexType>
    <xs:sequence>

      <xs:element name="raidArray" minOccurs="0" maxOccurs="unbounded">
        <xs:complexType>
          <xs:sequence>

            <xs:element name="level" type="raidLevel"/>

            <xs:element name="raidVolume" maxOccurs="unbounded">
              <xs:complexType>
                <xs:sequence>

                  <xs:element name="stripeSize" type="stripeSize"/>

                  <xs:element name="cachePolicy" type="cachePolicy"/>

                  <xs:element name="readPolicy" type="readPolicy"/>

                  <xs:element name="writePolicy" type="writePolicy"/>

                  <xs:element name="size" type="volumeSize"/>

                </xs:sequence>
              </xs:complexType>
            </xs:element>

            <xs:choice>

              <xs:element name="device" maxOccurs="unbounded">
                <xs:complexType>
                  <xs:sequence>

                    <xs:element name="enclosureID" type="enclosureID"/>

                    <xs:element name="slotNumber" type="slotNumber"/>

                  </xs:sequence>
                </xs:complexType>
              </xs:element>

              <xs:element name="span" minOccurs="2" maxOccurs="8">
                <xs:complexType>
                  <xs:sequence>

                    <xs:element name="device" maxOccurs="unbounded">
                      <xs:complexType>
                        <xs:sequence>
```

```
                        <xs:element name="enclosureID" type="enclosureID"/>

                        <xs:element name="slotNumber" type="slotNumber"/>

                      </xs:sequence>
                    </xs:complexType>
                  </xs:element>

                </xs:sequence>
              </xs:complexType>
            </xs:element>

          </xs:choice>

        </xs:sequence>
      </xs:complexType>
    </xs:element>

  </xs:sequence>
 </xs:complexType>
</xs:element>

</xs:schema>
```

D.3 Example: Default Node Partitioning

The following example follows the schema definition of section D.1, and shows the default layout used for regular nodes:

Example

```
<?xml version="1.0" encoding="ISO-8859-1"?>

<diskSetup xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance">
  <device>
    <blockdev>/dev/sda</blockdev>
    <blockdev>/dev/hda</blockdev>
    <blockdev>/dev/vda</blockdev>
    <blockdev>/dev/xvda</blockdev>
    <blockdev>/dev/cciss/c0d0</blockdev>
    <blockdev>/dev/nvme0n1</blockdev>
    <!-- the following is a by-path reference example -->
    <blockdev>/dev/disk/by-path/pci-0000:03:00.0-scsi-0:3:111:0</blockdev>
    <blockdev mode="cloud">/dev/sdb</blockdev>
    <blockdev mode="cloud">/dev/hdb</blockdev>
    <blockdev mode="cloud">/dev/vdb</blockdev>
    <blockdev mode="cloud">/dev/xvdb</blockdev>
    <!-- the following for paravirtual rhel6: -->
    <blockdev mode="cloud">/dev/xvdf</blockdev>
    <!-- the following for nvme volumes -->
    <blockdev mode="cloud">/dev/nvme1n1</blockdev>

    <partition id="a0" partitiontype="esp">
      <size>100M</size>
      <type>linux</type>
      <filesystem>fat</filesystem>
```

```
      <mountPoint>/boot/efi</mountPoint>
      <mountOptions>defaults,noatime,nodiratime</mountOptions>
    </partition>

    <partition id="a1">
      <size>20G</size>
      <type>linux</type>
      <filesystem>xfs</filesystem>
      <mountPoint>/</mountPoint>
      <mountOptions>defaults,noatime,nodiratime</mountOptions>
    </partition>

    <partition id="a2">
      <size>6G</size>
      <type>linux</type>
      <filesystem>xfs</filesystem>
      <mountPoint>/var</mountPoint>
      <mountOptions>defaults,noatime,nodiratime</mountOptions>
    </partition>

    <partition id="a3">
      <size>2G</size>
      <type>linux</type>
      <filesystem>xfs</filesystem>
      <mountPoint>/tmp</mountPoint>
      <mountOptions>defaults,noatime,nodiratime,nosuid,nodev</mountOptions>
    </partition>

    <partition id="a4">
      <size>12G</size>
      <type>linux swap</type>
    </partition>

    <partition id="a5">
      <size>max</size>
      <type>linux</type>
      <filesystem>xfs</filesystem>
      <mountPoint>/local</mountPoint>
      <mountOptions>defaults,noatime,nodiratime</mountOptions>
    </partition>

  </device>
</diskSetup>
```

The following sections (D.3.1–D.3.5) elaborate on aspects of this example.

D.3.1 Multiple Disks And Devices

The example assumes a single disk. Because multiple `blockdev` tags are used, the node-installer first tries to use `/dev/sda`, then `/dev/hda`, then `/dev/vda` (virtio disks), then `/dev/xvda` (xen disks), and so on. Another disk can be added by adding another pair of `<device></device>` tags and filling in the partitioning specifications for the next disk.

D.3.2 Persistent Naming For NVMe Disks

From BCM point release 11.34.0 onwards, persistent udev rules for NVMe disks can also be generated automatically by `bcm-generate-disk-udev-rules`, a script shipped as part of the `cmdaemon` pack-

age at /cm/local/apps/cmd/etc/htdocs/scripts/initialize/bcm-generate-disk-udev-rules, together with a matching finalize script at /cm/local/apps/cmd/etc/htdocs/scripts/finalize/bcm-generate-disk-udev-rules-finalize. The script is applied as an initialize/finalize script pair on a category, typically the category used for DGX nodes, rather than being run standalone. Persistent rules are generated from each NVMe disk's PCI address, and the disks are named osdisk-<*N*> and raiddisk-<*N*>.

This approach assumes that the NVMe disks in the system fall into exactly two size classes: the smaller-capacity disks are named osdisk-1, osdisk-2, and so on, while the larger-capacity disks are named raiddisk-1, raiddisk-2, and so on. Where this assumption does not hold, or where more control over the resulting names is needed, persistent udev rules can instead be created by hand, as described in a Knowledge Article at https://enterprise-support.nvidia.com/s/article/create-persistent-udev-rules-to-rename-the-disks-consistently-based-on-hardware-address. Once the udev rules exist, whichever method was used to create them, the resulting names (for example /dev/osdisk-1) can be used directly as <blockdev> values in a disk layout, in the same way that a by-path device reference can be used, as described next.

D.3.3 By-Path Device References And Cloud Devices

A blockdev tag can also specify a by-path device reference, for example /dev/disk/by-path/pci-0000:03:00.0-scsi-0:3:111:0, instead of a device path such as /dev/sda. These are udev-generated symlinks, named after each device's PCI/SCSI bus location, that point to the corresponding kernel device name. The bus location, and therefore the by-path name, is normally stable across reboots, though it is not absolutely guaranteed: boot-time enumeration timing can occasionally shift it even without any hardware change. Using a by-path reference is useful when a node's device name (for example /dev/nvme0n1) is not guaranteed to stay persistent across boots, such as when provisioning a matched pair of head nodes that must end up with an identical disk layout. A node's by-path names can be found by running ls -l /dev/disk/by-path on that node:

```
# ls -l /dev/disk/by-path/
total 0
lrwxrwxrwx 1 root root  9 Oct 19 13:51 pci-0000:03:00.0-scsi-0:2:3:0 -> ../../sda
lrwxrwxrwx 1 root root  9 Oct 19 13:51 pci-0000:03:00.0-scsi-0:2:4:0 -> ../../sdb
lrwxrwxrwx 1 root root  9 Oct 19 13:51 pci-0000:03:00.0-scsi-0:2:5:0 -> ../../sdd
lrwxrwxrwx 1 root root 10 Oct 19 13:51 pci-0000:03:00.0-scsi-0:2:5:0-part1 -> ../../sdd1
lrwxrwxrwx 1 root root 10 Oct 19 13:51 pci-0000:03:00.0-scsi-0:2:5:0-part2 -> ../../sdd2
lrwxrwxrwx 1 root root 10 Oct 19 13:51 pci-0000:03:00.0-scsi-0:2:5:0-part3 -> ../../sdd3
lrwxrwxrwx 1 root root  9 Oct 19 13:51 pci-0000:03:00.0-scsi-0:2:6:0 -> ../../sdc
lrwxrwxrwx 1 root root  9 Oct 19 13:51 pci-0000:03:00.0-scsi-0:2:7:0 -> ../../sdg
lrwxrwxrwx 1 root root  9 Oct 19 13:51 pci-0000:03:00.0-scsi-0:2:8:0 -> ../../sdf
lrwxrwxrwx 1 root root  9 Oct 19 13:51 pci-0000:03:00.0-scsi-0:3:111:0 -> ../../sde
lrwxrwxrwx 1 root root  9 Oct 19 13:51 pci-0000:61:00.4-usb-0:2:1.0-scsi-0:0:0:0 -> ../../sr0
```

Each entry maps a stable PCI path name to a kernel device name; the left-hand name can be used as the <blockdev> value.

Cloud devices can also be accessed using the mode="cloud" option. Removing block devices from the layout if they are not going to be used does no harm.

D.3.4 Partition Sizing And Filesystem Types

For each partition, a size tag is specified. Sizes can be specified using megabytes (e.g. 500M), gigabytes (e.g. 50G) or terabytes (e.g. 2T or 4.5T). Relative sizes, without units, can be used in the form of fractions (e.g. 2/3) or percentages (e.g. 70%), which can be useful for disk sizes that are not known in advance.

Small differences in size do not trigger a full install for existing relative partitions.

For swap partitions, a size of auto sets a swap partition to twice the node memory size. If there is more than one swap partition, then the priority tag can be set so that the partition with the higher

priority is used first.

For a device, the attribute `max` for a `size` tag forces the last device in the partition to use all remaining space, and if needed, adjusts the implementation of the sequence of `size` tags in the remaining partitions accordingly. The use of `max` for a partition is convenient.

In the example, all non-boot filesystems are specified as `xfs`. One of the valid alternatives is `ext4`.

D.3.5 Mount Options

The `mount` man page has more details on mount options. If the `mountOptions` tag is left empty, its value defaults to `defaults`.

D.4 Example: Hardware RAID Configuration

A prerequisite with hardware RAID is that it must be enabled and configured properly in the BIOS.

If it is enabled and configured correctly, then the hardware RAID configuration can be defined or modified by setting the `hardwareraidconfiguration` parameter in `device` or `category` mode:

Example

```
[root@basecm11 ~]# cmsh
[basecm11]% category use default
[basecm11->category[default]]% set hardwareraidconfiguration
```

This opens up an editor in which the XML file can be specified according to the schema in section D.2. XML validation is carried out.

D.4.1 RAID level 0 And RAID 10 Example

In the following configuration the node has two RAID arrays, one in a RAID 0 and the other in a RAID 10 configuration:

- The RAID 0 array contains three volumes and is made up of two hard disks, placed in slots 0 and 1. The volumes have different values for the options and policies.
- The RAID 10 array consists of just one volume and has two spans, in slots 2 and 3. Each span has two hard disks.

Example

```
<raidSetup>
  <raidArray>
    <level>0</level>

    <raidVolume>
      <stripeSize>64K</stripeSize>
      <cachePolicy>Direct</cachePolicy>
      <readPolicy>NORA</readPolicy>
      <writePolicy>WT</writePolicy>
      <size>40G</size>
    </raidVolume>

    <raidVolume>
      <stripeSize>128K</stripeSize>
      <cachePolicy>Direct</cachePolicy>
      <readPolicy>RA</readPolicy>
      <writePolicy>WB</writePolicy>
      <size>80G</size>
```

```
    </raidVolume>

    <raidVolume>
      <stripeSize>256K</stripeSize>
      <cachePolicy>Cached</cachePolicy>
      <readPolicy>ADRA</readPolicy>
      <writePolicy>WT</writePolicy>
      <size>100G</size>
    </raidVolume>

    <device>
      <enclosureID>auto</enclosureID>
      <slotNumber>0</slotNumber>
    </device>

    <device>
      <enclosureID>32</enclosureID>
      <slotNumber>1</slotNumber>
    </device>
  </raidArray>

  <raidArray>
    <level>10</level>

    <raidVolume>
      <stripeSize>64K</stripeSize>
      <cachePolicy>Direct</cachePolicy>
      <readPolicy>NORA</readPolicy>
      <writePolicy>WT</writePolicy>
      <size>40G</size>
    </raidVolume>

    <span>
      <device>
        <enclosureID>auto</enclosureID>
        <slotNumber>2</slotNumber>
      </device>

      <device>
        <enclosureID>auto</enclosureID>
        <slotNumber>3</slotNumber>
      </device>
    </span>

    <span>
      <device>
        <enclosureID>auto</enclosureID>
        <slotNumber>4</slotNumber>
      </device>

      <device>
        <enclosureID>auto</enclosureID>
        <slotNumber>5</slotNumber>
      </device>
    </span>
```

```
  </raidArray>
</raidSetup>
```

D.5 Example: Software RAID

The following example shows a simple software RAID setup:

Example

```
<?xml version="1.0" encoding="ISO-8859-1"?>

<diskSetup xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
           xsi:noNamespaceSchemaLocation="schema.xsd">

  <device>
    <blockdev>/dev/sda</blockdev>
    <partition id="a1">
      <size>25G</size>
      <type>linux raid</type>
    </partition>
  </device>

  <device>
    <blockdev>/dev/sdb</blockdev>
    <partition id="b1">
      <size>25G</size>
      <type>linux raid</type>
    </partition>
  </device>

  <raid id="r1">
    <member>a1</member>
    <member>b1</member>
    <level>1</level>
    <filesystem>ext3</filesystem>
    <mountPoint>/</mountPoint>
    <mountOptions>defaults,noatime,nodiratime</mountOptions>
  </raid>

</diskSetup>
```

The `level` tag specifies the RAID level used. The following are supported:

- `0` (striping without parity)
- `1` (mirroring)
- `4` (striping with dedicated parity drive)
- `5` (striping with distributed parity)
- `6` (striping with distributed double parity)
- `10` (mirroring and striping)

The `member` tags must refer to an `id` attribute of a `partition` tag, or an `id` attribute of a another `raid` tag. The latter can be used to create, for example, a nested RAID 10 (1+0) configuration, instead of a complex RAID 10 configuration as specified by a `<level>10</level>` entry. A good explanation of the

difference between complex and nested RAID 10 can be found at `https://documentation.suse.com/sles/15-SP5/html/SLES-all/cha-raid10.html#sec-raid10-complex`.

The administrator must ensure that the correct RAID kernel module is loaded (section 5.3.2). Including the appropriate module from the following is usually sufficient: `raid0`, `raid1`, `raid4`, `raid5`, `raid6`, `raid10`.

D.6 Example: Software RAID With Swap

The `<swap></swap>` tag is used to indicate a swap partition in a RAID device specified in the XML schema of section D.1. For example, the following marks a 1GB RAID 1 partition as being used for swap, and the second partition for an ext3 filesystem:

```
<?xml version="1.0" encoding="UTF-8"?>

<diskSetup xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance">
  <device>
    <blockdev>/dev/sda</blockdev>
    <partition id="a1">
      <size>1G</size>
      <type>linux raid</type>
    </partition>
    <partition id="a2">
      <size>max</size>
      <type>linux raid</type>
    </partition>
  </device>
  <device>
    <blockdev>/dev/sdb</blockdev>
    <partition id="b1">
      <size>1G</size>
      <type>linux raid</type>
    </partition>
    <partition id="b2">
      <size>max</size>
      <type>linux raid</type>
    </partition>
  </device>
  <raid id="r1">
    <member>a1</member>
    <member>b1</member>
    <level>1</level>
    <swap></swap>
  </raid>
  <raid id="r2">
    <member>a2</member>
    <member>b2</member>
    <level>1</level>
    <filesystem>ext3</filesystem>
    <mountPoint>/</mountPoint>
    <mountOptions>defaults,noatime,nodiratime</mountOptions>
  </raid>
</diskSetup>
```

As in section D.5, the appropriate RAID modules must be loaded beforehand.

D.7 Example: Logical Volume Manager

This example shows a simple LVM setup:

Example

```
<?xml version="1.0" encoding="UTF-8"?>
<diskSetup xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance">
  <device>
    <blockdev>/dev/sda</blockdev>
    <blockdev>/dev/hda</blockdev>
    <blockdev>/dev/vda</blockdev>
    <blockdev>/dev/xvda</blockdev>
    <blockdev>/dev/cciss/c0d0</blockdev>
    <blockdev mode="cloud">/dev/sdb</blockdev>
    <blockdev mode="cloud">/dev/hdb</blockdev>
    <blockdev mode="cloud">/dev/vdb</blockdev>
    <blockdev mode="cloud">/dev/xvdb</blockdev>
    <blockdev mode="cloud">/dev/xvdf</blockdev>
    <partition id="a1">
      <size>1G</size>
      <type>linux</type>
      <filesystem>ext2</filesystem>
      <mountPoint>/boot</mountPoint>
      <mountOptions>defaults,noatime,nodiratime</mountOptions>
    </partition>
    <partition id="a2">
      <size>16G</size>
      <type>linux swap</type>
    </partition>
    <partition id="a3">
      <size>max</size>
      <type>linux lvm</type>
    </partition>
  </device>
  <volumeGroup>
    <name>vg01</name>
    <extentSize>8M</extentSize>
    <physicalVolumes>
      <member>a3</member>
    </physicalVolumes>
    <logicalVolumes>
      <volume>
        <name>lv00</name>
        <size>max</size>
        <filesystem>ext3</filesystem>
        <mountPoint>/</mountPoint>
        <mountOptions>defaults,noatime,nodiratime</mountOptions>
      </volume>
      <volume>
        <name>lv01</name>
        <size>8G</size>
        <filesystem>ext3</filesystem>
        <mountPoint>/tmp</mountPoint>
        <mountOptions>defaults,noatime,nodiratime</mountOptions>
      </volume>
    </logicalVolumes>
```

```
  </volumeGroup>
</diskSetup>
```

The `member` tags must refer to an `id` attribute of a `partition` tag, or an `id` attribute of a `raid` tag.

The administrator must ensure that the `dm-mod` kernel module is loaded when LVM is used.

D.8 Example: Logical Volume Manager With RAID 1

This example shows an LVM setup, but with the LVM partitions mirrored using RAID 1:

Example

```
<?xml version="1.0" encoding="ISO-8859-1"?>
<diskSetup xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
           xsi:noNamespaceSchemaLocation="schema.xsd">

<device>
  <blockdev>/dev/sda</blockdev>
    <partition id="a1">
      <size>1G</size>
      <type>linux raid</type>
    </partition>
    <partition id="a2">
      <size>max</size>
      <type>linux raid</type>
    </partition>
</device>

<device>
  <blockdev>/dev/sdb</blockdev>
    <partition id="b1">
      <size>1G</size>
      <type>linux raid</type>
    </partition>
    <partition id="b2">
      <size>max</size>
      <type>linux raid</type>
    </partition>
</device>

<raid id="r1">
  <member>a1</member>
  <member>b1</member>
  <level>1</level>
  <filesystem>ext3</filesystem>
  <mountPoint>/boot</mountPoint>
  <mountOptions>defaults,noatime,nodiratime</mountOptions>
</raid>

<raid id="r2">
  <member>a2</member>
  <member>b2</member>
  <level>1</level>
</raid>

<volumeGroup>
```

```
  <name>vg01</name>
  <extentSize>8M</extentSize>
  <physicalVolumes>
    <member>r2</member>
  </physicalVolumes>

  <logicalVolumes>
    <volume>
      <name>lv00</name>
      <size>50G</size>
      <filesystem>ext3</filesystem>
      <mountPoint>/</mountPoint>
      <mountOptions>defaults,noatime,nodiratime</mountOptions>
    </volume>
    <volume>
      <name>lv01</name>
      <size>25G</size>
      <filesystem>ext3</filesystem>
      <mountPoint>/tmp</mountPoint>
      <mountOptions>defaults,noatime,nodiratime</mountOptions>
    </volume>
    <volume>
      <name>lv02</name>
      <size>25G</size>
      <filesystem>ext3</filesystem>
      <mountPoint>/var</mountPoint>
      <mountOptions>defaults,noatime,nodiratime</mountOptions>
    </volume>
  </logicalVolumes>
</volumeGroup>

</diskSetup>
```

D.9 Example: Diskless

This example shows a node configured for diskless operation:

Example

```
<?xml version="1.0" encoding="UTF-8"?>

<diskSetup xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance">
  <diskless maxMemSize="90%"></diskless>
</diskSetup>
```

An example of the implementation of a diskless configuration is given in section 3.13.3.

In diskless mode the software image is transferred by the node-installer to a RAM-based filesystem on the node called `tmpfs`.

The obvious advantage of running from RAM is the elimination of the physical disk, cutting power consumption and reducing the chance of hardware failure. On the other hand, some of the RAM on the node is then no longer available for user applications.

Special considerations with diskless mode:

- **Recommended minimum RAM size:** The available RAM per node should be sufficient to run the OS and the required tasks. At least 4GB is recommended for diskless nodes.

- **The `tmpfs` size limit:** The maximum amount of RAM that can be used for a filesystem is set with the `maxMemSize` attribute. A value of `100%` allows all of the RAM to be used. The default value is `90%`. A value of `0`, without the % sign, removes all restrictions.

  A limit does not however necessarily prevent the node from crashing, as some processes might not deal properly with a situation when there is no more space left on the filesystem.

- **Persistence issues:** While running as a diskless node, the node is unable to retain any non-shared data each time it reboots. For example the files in `/var/log/*`, which are normally preserved by the exclude list settings for disked nodes, are lost from RAM during diskless mode reboots. The installmode `NOSYNC` setting cannot be used with diskless nodes during a node reboot.

- **Leftover disk issues:** Administrators in charge of sensitive environments should be aware that the disk of a node that is now running in diskless mode still contains files from the last time the disk was used, unless the files are explicitly wiped.

- **Reducing the software image size in `tmpfs` on a diskless node:** To make more RAM available for tasks, the software image size held in RAM can be reduced:

  – by removing unnecessary software from the image.

  – by mounting parts of the filesystem in the image over NFS during normal use. This is especially worthwhile for less frequently accessed parts of the image (section 3.14.3).

D.10 Example: Semi-diskless

Diskless operation (section D.9) can also be mixed with certain parts of the filesystem on the local physical disk. This frees up RAM which the node can then put to other use. In this example all data in `/local` is on the physical disk, the rest in RAM.

Example

```
<?xml version="1.0" encoding="UTF-8"?>

<diskSetup xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
           xsi:noNamespaceSchemaLocation="schema.xsd">
  <diskless maxMemSize="0"></diskless>
  <device>
    <blockdev>/dev/sda</blockdev>
    <partition id="a1">
      <size>max</size>
      <type>linux</type>
      <filesystem>ext3</filesystem>
      <mountPoint>/local</mountPoint>
      <mountOptions>defaults,noatime,nodiratime</mountOptions>
    </partition>
  </device>
</diskSetup>
```

When nodes operate in semi-diskless mode the node-installer always uses `excludelistfullinstall` (section 5.4.7) when synchronizing the software image to memory and disk.

An alternative to using a local disk for freeing up RAM is to use NFS storage, as is described in section 3.14.3.

D.11 Example: Preventing Accidental Data Loss

Optional tags, `vendor` and `requiredSize`, can be used to prevent accidentally repartitioning the wrong drive. Such a tag use is shown in the following example.

Example

```
<?xml version="1.0" encoding="ISO-8859-1"?>

<diskSetup xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
           xsi:noNamespaceSchemaLocation="schema.xsd">
  <device>
    <blockdev>/dev/sda</blockdev>
    <vendor>Hitachi</vendor>
    <requiredSize>200G</requiredSize>

    <partition id="a1">
      <size>max</size>
      <type>linux</type>
      <filesystem>ext3</filesystem>
      <mountPoint>/</mountPoint>
      <mountOptions>defaults,noatime,nodiratime</mountOptions>
    </partition>

  </device>
  <device>
    <blockdev>/dev/sdb</blockdev>
    <vendor>BigRaid</vendor>
    <requiredSize>2T</requiredSize>

    <partition id="b1">
      <size>max</size>
      <type>linux</type>
      <filesystem>ext3</filesystem>
      <mountPoint>/data</mountPoint>
      <mountOptions>defaults,noatime,nodiratime</mountOptions>
    </partition>

  </device>
</diskSetup>
```

If a `vendor` or a `requiredSize` element is specified, it is treated as an assertion which is checked by the node-installer. The node-installer reads the drive vendor string from `/sys/block/`*<drive name>*`/device/vendor`. For the assertion to succeed, the ratio of actual disk size to the value specified by `requiredSize`, should be at least 0.85:1, and at most 1:0.85.

That is: to be able to get past the `requiredSize` assertion, the actual drive size as seen from `fdisk -l` should be 85% to about 118% of the asserted size.

If any assertion fails, no partitioning changes will be made to any of the specified devices.

For assertions with drives that are similar or identical in size, and are from the same vendor, the `requiredSize` and `vendor` elements are not enough to differentiate between the drives. In such cases, custom assertions (section D.12) can be set for particular drives.

Specifying device assertions is recommended for nodes that contain important data because it protects against a situation where a drive is assigned to an incorrect block device. This can happen, for example, when the first drive, for example `/dev/sda`, in a multi-drive system is not detected (e.g. due to a hardware failure, or a BIOS update) which could cause the second drive to become known as `/dev/sda`, potentially causing much woe.

As an aside, CMDaemon does offer another way, outside of assertions, to avoid wiping out drive data automatically. This is done in `cmsh` by setting the boolean value of `datanode` to `yes` (section 5.4.4).

D.12 Example: Using Custom Assertions

The following example shows the use of the `assert` tag, which can be added to a `device` definition:

Example

```
<?xml version="1.0" encoding="ISO-8859-1"?>

<diskSetup xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
           xsi:noNamespaceSchemaLocation="schema.xsd">
  <device>
    <blockdev>/dev/sda</blockdev>
    <assert name="modelCheck" args="WD800AAJS">
      <![CDATA[
        #!/bin/bash
        if grep -q $1 /sys/block/$ASSERT_DEV/device/model; then
          exit 0
        else
          exit 1
        fi
      ]]>
    </assert>

    <partition id="a1">
      <size>max</size>
      <type>linux</type>
      <filesystem>ext3</filesystem>
      <mountPoint>/</mountPoint>
      <mountOptions>defaults,noatime,nodiratime</mountOptions>
    </partition>

  </device>
  <device>
    <blockdev>/dev/sdb</blockdev>
    <vendor>BigRaid</vendor>
    <requiredSize>2T</requiredSize>

    <partition id="b1">
      <size>max</size>
      <type>linux</type>
      <filesystem>ext3</filesystem>
      <mountPoint>/data</mountPoint>
      <mountOptions>defaults,noatime,nodiratime</mountOptions>
    </partition>

  </device>
</diskSetup>
```

The `assert` tag is similar to the `vendor` and `size` tags described in section D.11.

It can be used to define custom assertions. The assertions can be implemented using any script language.

The script can access the environment variables `ASSERT_DEV` (eg: `sda`) and `ASSERT_NODE` (eg: `/dev/sda`) during the node-installer stage.

Each assert needs to be assigned an arbitrary name and can be passed custom parameters. A non-zero exit code in the assertion causes the node-installer to halt.

D.13 Example: Software RAID1 With One Big Partition

The following example shows a head node hard drive that uses one big partition with software RAID 1.

Example

```
<?xml version="1.0" encoding="ISO-8859-1"?>

<diskSetup xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance">
  <device>
    <blockdev>/dev/sda</blockdev>
    <blockdev>/dev/hda</blockdev>
    <blockdev>/dev/vda</blockdev>
    <blockdev>/dev/xvda</blockdev>
    <blockdev>/dev/cciss/c0d0</blockdev>
    <blockdev>/dev/nvme0n1</blockdev>
    <partition id="a0" partitiontype="esp">
      <size>100M</size>
      <type>linux</type>
      <filesystem>fat</filesystem>
      <mountPoint>/boot/efi</mountPoint>
      <mountOptions>defaults,noatime,nodiratime</mountOptions>
    </partition>
    <partition id="a1">
      <size>1G</size>
      <type>linux raid</type>
    </partition>
    <partition id="a2">
      <size>16G</size>
      <type>linux raid</type>
    </partition>
    <partition id="a3">
      <size>max</size>
      <type>linux raid</type>
    </partition>
  </device>

  <device>
    <blockdev>/dev/sdb</blockdev>
    <blockdev>/dev/hdb</blockdev>
    <blockdev>/dev/vdb</blockdev>
    <blockdev>/dev/xvdb</blockdev>
    <blockdev>/dev/cciss/c0d1</blockdev>
    <blockdev>/dev/nvme1n1</blockdev>
    <partition id="b0" partitiontype="esp">
      <size>100M</size>
      <type>linux</type>
      <filesystem>fat</filesystem>
    </partition>
    <partition id="b1">
      <size>1G</size>
      <type>linux raid</type>
    </partition>
    <partition id="b2">
      <size>16G</size>
      <type>linux raid</type>
    </partition>
```

```
    <partition id="b3">
      <size>max</size>
      <type>linux raid</type>
    </partition>
  </device>

  <raid id="r1">
    <member>a1</member>
    <member>b1</member>
    <level>1</level>
    <filesystem>ext2</filesystem>
    <mountPoint>/boot</mountPoint>
    <mountOptions>defaults,noatime,nodiratime</mountOptions>
  </raid>

  <raid id="r2">
    <member>a2</member>
    <member>b2</member>
    <level>1</level>
    <swap/>
  </raid>

  <raid id="r3">
    <member>a3</member>
    <member>b3</member>
    <level>1</level>
    <filesystem>__FSTYPE__</filesystem>
    <mountPoint>/</mountPoint>
    <mountOptions>defaults,noatime,nodiratime</mountOptions>
  </raid>
</diskSetup>
```

D.14 Example: Software RAID5 With One Big Partition

The following example shows a head node hard drive that uses one big partition with software RAID 5.

Example

```
<?xml version="1.0" encoding="ISO-8859-1"?>

<diskSetup xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance">
  <device>
    <blockdev>/dev/sda</blockdev>
    <blockdev>/dev/hda</blockdev>
    <blockdev>/dev/vda</blockdev>
    <blockdev>/dev/xvda</blockdev>
    <blockdev>/dev/cciss/c0d0</blockdev>
    <blockdev>/dev/nvme0n1</blockdev>
    <partition id="a0" partitiontype="esp">
      <size>100M</size>
      <type>linux</type>
      <filesystem>fat</filesystem>
      <mountPoint>/boot/efi</mountPoint>
      <mountOptions>defaults,noatime,nodiratime</mountOptions>
    </partition>
    <partition id="a1">
```

```
    <size>1G</size>
    <type>linux raid</type>
  </partition>
  <partition id="a2">
    <size>16G</size>
    <type>linux raid</type>
  </partition>
  <partition id="a3">
    <size>max</size>
    <type>linux raid</type>
  </partition>
</device>

<device>
  <blockdev>/dev/sdb</blockdev>
  <blockdev>/dev/hdb</blockdev>
  <blockdev>/dev/vdb</blockdev>
  <blockdev>/dev/xvdb</blockdev>
  <blockdev>/dev/cciss/c0d1</blockdev>
  <blockdev>/dev/nvme1n1</blockdev>
  <partition id="b0" partitiontype="esp">
    <size>100M</size>
    <type>linux</type>
    <filesystem>fat</filesystem>
  </partition>
  <partition id="b1">
    <size>1G</size>
    <type>linux raid</type>
  </partition>
  <partition id="b2">
    <size>16G</size>
    <type>linux raid</type>
  </partition>
  <partition id="b3">
    <size>max</size>
    <type>linux raid</type>
  </partition>
</device>

<device>
  <blockdev>/dev/sdc</blockdev>
  <blockdev>/dev/hdc</blockdev>
  <blockdev>/dev/vdc</blockdev>
  <blockdev>/dev/xvdc</blockdev>
  <blockdev>/dev/cciss/c0d2</blockdev>
  <blockdev>/dev/nvme2n1</blockdev>
  <partition id="c0" partitiontype="esp">
    <size>100M</size>
    <type>linux</type>
    <filesystem>fat</filesystem>
  </partition>
  <partition id="c1">
    <size>1G</size>
    <type>linux raid</type>
  </partition>
```

```
    <partition id="c2">
      <size>16G</size>
      <type>linux raid</type>
    </partition>
    <partition id="c3">
      <size>max</size>
      <type>linux raid</type>
    </partition>
  </device>

  <raid id="r1">
    <member>a1</member>
    <member>b1</member>
    <member>c1</member>
    <level>1</level>
    <filesystem>ext2</filesystem>
    <mountPoint>/boot</mountPoint>
    <mountOptions>defaults,noatime,nodiratime</mountOptions>
  </raid>

  <raid id="r2">
    <member>a2</member>
    <member>b2</member>
    <member>c2</member>
    <level>5</level>
    <swap/>
  </raid>

  <raid id="r3">
    <member>a3</member>
    <member>b3</member>
    <member>c3</member>
    <level>5</level>
    <filesystem>__FSTYPE__</filesystem>
    <mountPoint>/</mountPoint>
    <mountOptions>defaults,noatime,nodiratime</mountOptions>
  </raid>
</diskSetup>
```

D.15 Example: Software RAID1 With Standard Partitioning

The following example shows a head node hard drive that uses the standard partitioning with software RAID 1.

Example

```
<?xml version="1.0" encoding="ISO-8859-1"?>

<diskSetup xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance">
  <device>
    <blockdev>/dev/sda</blockdev>
    <blockdev>/dev/hda</blockdev>
    <blockdev>/dev/vda</blockdev>
    <blockdev>/dev/xvda</blockdev>
    <blockdev>/dev/cciss/c0d0</blockdev>
    <blockdev>/dev/nvme0n1</blockdev>
```

```
    <partition id="a0" partitiontype="esp">
      <size>100M</size>
      <type>linux</type>
      <filesystem>fat</filesystem>
      <mountPoint>/boot/efi</mountPoint>
      <mountOptions>defaults,noatime,nodiratime</mountOptions>
    </partition>

    <partition id="a1">
      <size>1G</size>
      <type>linux raid</type>
    </partition>

    <partition id="a2">
      <size>16G</size>
      <type>linux raid</type>
    </partition>

    <partition id="a3">
      <size>8G</size>
      <type>linux raid</type>
    </partition>

    <partition id="a4">
      <size>30G</size>
      <type>linux raid</type>
    </partition>

    <partition id="a5">
      <size>max</size>
      <type>linux raid</type>
    </partition>
  </device>

  <device>
    <blockdev>/dev/sdb</blockdev>
    <blockdev>/dev/hdb</blockdev>
    <blockdev>/dev/vdb</blockdev>
    <blockdev>/dev/xvdb</blockdev>
    <blockdev>/dev/cciss/c0d1</blockdev>
    <blockdev>/dev/nvme1n1</blockdev>

    <partition id="b0" partitiontype="esp">
      <size>100M</size>
      <type>linux</type>
      <filesystem>fat</filesystem>
    </partition>

    <partition id="b1">
      <size>1G</size>
      <type>linux raid</type>
    </partition>

    <partition id="b2">
```

```
    <size>16G</size>
    <type>linux raid</type>
  </partition>

  <partition id="b3">
    <size>8G</size>
    <type>linux raid</type>
  </partition>

  <partition id="b4">
    <size>30G</size>
    <type>linux raid</type>
  </partition>

  <partition id="b5">
    <size>max</size>
    <type>linux raid</type>
  </partition>
</device>

<raid id="r1">
  <member>a1</member>
  <member>b1</member>
  <level>1</level>
  <filesystem>ext2</filesystem>
  <mountPoint>/boot</mountPoint>
  <mountOptions>defaults,noatime,nodiratime</mountOptions>
</raid>

<raid id="r2">
  <member>a2</member>
  <member>b2</member>
  <level>1</level>
  <swap/>
</raid>

<raid id="r3">
  <member>a3</member>
  <member>b3</member>
  <level>1</level>
  <filesystem>__FSTYPE__</filesystem>
  <mountPoint>/tmp</mountPoint>
  <mountOptions>defaults,noatime,nodiratime,nosuid,nodev</mountOptions>
</raid>

<raid id="r4">
  <member>a4</member>
  <member>b4</member>
  <level>1</level>
  <filesystem>__FSTYPE__</filesystem>
  <mountPoint>/var</mountPoint>
  <mountOptions>defaults,noatime,nodiratime</mountOptions>
</raid>

<raid id="r5">
```

```
    <member>a5</member>
    <member>b5</member>
    <level>1</level>
    <filesystem>__FSTYPE__</filesystem>
    <mountPoint>/</mountPoint>
    <mountOptions>defaults,noatime,nodiratime</mountOptions>
  </raid>

</diskSetup>
```

D.16 Example: Software RAID5 With Standard Partitioning

The following example shows a head node hard drive that uses the standard partitioning with software RAID 5.

Example

```
<?xml version="1.0" encoding="ISO-8859-1"?>
<diskSetup xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance">
  <device>
    <blockdev>/dev/sda</blockdev>
    <blockdev>/dev/hda</blockdev>
    <blockdev>/dev/vda</blockdev>
    <blockdev>/dev/xvda</blockdev>
    <blockdev>/dev/cciss/c0d0</blockdev>
    <blockdev>/dev/nvme0n1</blockdev>
    <partition id="a0" partitiontype="esp">
      <size>100M</size>
      <type>linux</type>
      <filesystem>fat</filesystem>
      <mountPoint>/boot/efi</mountPoint>
      <mountOptions>defaults,noatime,nodiratime</mountOptions>
    </partition>

    <partition id="a1">
      <size>1G</size>
      <type>linux raid</type>
    </partition>
    <partition id="a2">
      <size>6G</size>
      <type>linux raid</type>
    </partition>
    <partition id="a3">
      <size>8G</size>
      <type>linux raid</type>
    </partition>
    <partition id="a4">
      <size>30G</size>
      <type>linux raid</type>
    </partition>
    <partition id="a5">
      <size>max</size>
      <type>linux raid</type>
    </partition>
  </device>
  <device>
```

```
    <blockdev>/dev/sdb</blockdev>
    <blockdev>/dev/hdb</blockdev>
    <blockdev>/dev/vdb</blockdev>
    <blockdev>/dev/xvdb</blockdev>
    <blockdev>/dev/cciss/c0d1</blockdev>
    <blockdev>/dev/nvme1n1</blockdev>
    <partition id="b0" partitiontype="esp">
      <size>100M</size>
      <type>linux</type>
      <filesystem>fat</filesystem>
    </partition>
    <partition id="b1">
      <size>1G</size>
      <type>linux raid</type>
    </partition>
    <partition id="b2">
      <size>6G</size>
      <type>linux raid</type>
    </partition>
    <partition id="b3">
      <size>8G</size>
      <type>linux raid</type>
    </partition>
    <partition id="b4">
      <size>30G</size>
      <type>linux raid</type>
    </partition>
    <partition id="b5">
      <size>max</size>
      <type>linux raid</type>
    </partition>
  </device>
  <device>
    <blockdev>/dev/sdc</blockdev>
    <blockdev>/dev/hdc</blockdev>
    <blockdev>/dev/vdc</blockdev>
    <blockdev>/dev/xvdc</blockdev>
    <blockdev>/dev/cciss/c0d2</blockdev>
    <blockdev>/dev/nvme2n1</blockdev>
    <partition id="c0" partitiontype="esp">
      <size>100M</size>
      <type>linux</type>
      <filesystem>fat</filesystem>
    </partition>
    <partition id="c1">
      <size>1G</size>
      <type>linux raid</type>
    </partition>
    <partition id="c2">
      <size>6G</size>
      <type>linux raid</type>
    </partition>
    <partition id="c3">
      <size>8G</size>
      <type>linux raid</type>
```

```
    </partition>
    <partition id="c4">
      <size>30G</size>
      <type>linux raid</type>
    </partition>
    <partition id="c5">
      <size>max</size>
      <type>linux raid</type>
    </partition>
  </device>
  <raid id="r1">
    <member>a1</member>
    <member>b1</member>
    <member>c1</member>
    <level>1</level>
    <filesystem>ext2</filesystem>
    <mountPoint>/boot</mountPoint>
    <mountOptions>defaults,noatime,nodiratime</mountOptions>
  </raid>
  <raid id="r2">
    <member>a2</member>
    <member>b2</member>
    <member>c2</member>
    <level>5</level>
    <swap/>
  </raid>
  <raid id="r3">
    <member>a3</member>
    <member>b3</member>
    <member>c3</member>
    <level>5</level>
    <filesystem>__FSTYPE__</filesystem>
    <mountPoint>/tmp</mountPoint>
    <mountOptions>defaults,noatime,nodiratime,nosuid,nodev</mountOptions>
  </raid>
  <raid id="r4">
    <member>a4</member>
    <member>b4</member>
    <member>c4</member>
    <level>5</level>
    <filesystem>__FSTYPE__</filesystem>
    <mountPoint>/var</mountPoint>
    <mountOptions>defaults,noatime,nodiratime</mountOptions>
  </raid>
  <raid id="r5">
    <member>a5</member>
    <member>b5</member>
    <member>c5</member>
    <level>5</level>
    <filesystem>__FSTYPE__</filesystem>
    <mountPoint>/</mountPoint>
    <mountOptions>defaults,noatime,nodiratime</mountOptions>
  </raid>
</diskSetup>
```

D.17 Example: LUKS Disk Encryption With Standard Partitioning

D.17.1 Introduction

This section considers XML configuration that uses LUKS to set encrypted partitions on a block device. Only the non-`boot` partitions can be encrypted.

Encryption can be configured for head and regular nodes during head node installation (section 3.3.16 of the *Installation Manual*). For regular nodes it can also be configured later on by modifying the XML file used to define disk layouts (section D.17.2).

LUKS disk encryption on nodes by default uses a passphrase to decrypt a partition. The passphrase processing can be handled automatically, or it can be handled by typing it in manually. These options are explained in greater detail next:

- Automatic passphrase processing: For a node that is configured for encryption, and that is provisioned over the network, the passphrase needs to be used to make the encrypted partition accessible. The acceptance of the passphrase for the node needs to be confirmed by the cluster administrator during boot. For security, similar to the familiar case of SSH confirmation for a new first-time connection, the cluster administrator should only confirm acceptance if sure that there is no man-in-the-middle attack.

  The cluster administrator needs only to confirm the passphrase to make the encrypted partition accessible. Other than confirmation, the cluster administrator does not need to directly manage or even know the passphrase, because the passphrase is stored and managed automatically via CMDaemon.

- Manual passphrase processing: A node that is not provisioned over the network—that is a node that is configured to boot from its own disk—is called a standalone node. Standalone nodes are generally discouraged in a BCM cluster because they are harder to manage, but there are use cases for them. For example, an edge director (section 2.1.7 of the *Edge Manual*) is configured as a standalone node by default, because it needs to have a degree of autonomy from the head node.

  If a standalone node—such as a head node or an edge director that is functioning autonomously—boots from its own drive, it may be that it is configured with an encrypted partition, due to security considerations. With such a configuration, the cluster administrator needs to be able to directly manage the passphrase to allow access to the encrypted partitions. So, after a node is configured to be a standalone node, access to an encrypted partition requires a passphrase to be entered from the console of that node when the node is booting from its drive.

D.17.2 Node Provisioned Over The Network: Encrypted Partition XML Example

XML Specification For An Encrypted Partition

The XML example that follows shows encrypted partitions set for a regular node that uses one big partition and one swap partition. The layout is based on a slightly modified version of the example XML file at:

```
/cm/local/apps/cmd/etc/htdocs/disk-setup/x86-64-slave-one-big-encrypted-partition-ext4.xml
```

with minimal `<encryption />` tags added for the root and swap partitions that are to run encrypted:

```
[basecm11->device[node001]]% get disksetup
<?xml version="1.0" encoding="ISO-8859-1"?>

<!-- One swap partition and the rest of the filesystem on 1 partition -->

<diskSetup xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance">
  <device>
    <blockdev>/dev/vdb</blockdev>
```

```
    <partition id="a0" partitiontype="esp">
      <size>100M</size>
      <type>linux</type>
      <filesystem>fat</filesystem>
      <mountPoint>/boot/efi</mountPoint>
      <mountOptions>defaults,noatime,nodiratime</mountOptions>
    </partition>

    <partition id="a1">
      <size>2G</size>
      <type>linux swap</type>
      <encryption />
    </partition>

    <partition id="a2">
      <size>max</size>
      <type>linux</type>
      <encryption />
      <filesystem>ext4</filesystem>
      <mountPoint>/</mountPoint>
      <mountOptions>defaults,noatime,nodiratime</mountOptions>
    </partition>
  </device>
</diskSetup>
```

Other XML file examples for encryption can found under `/cm/local/apps/cmd/etc/htdocs/disk-setup/`, with file names that start with the text `slave-one-big-encrypted-`.

Stages During The Provisioning Of An Encrypted Partition

After the configuration has been committed, and the node is rebooted, the node pauses during the boot process, and waits for approval of the passphrase for that node (figure D.1):

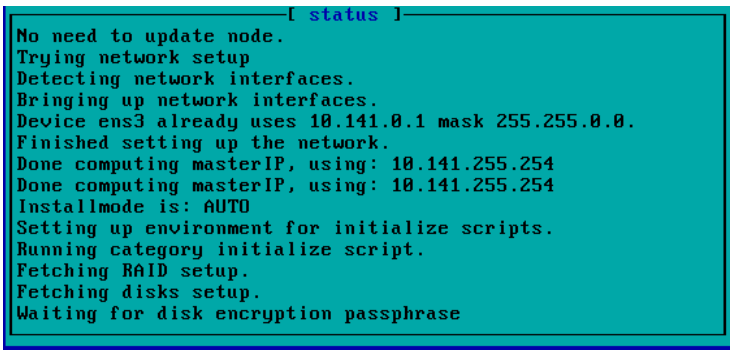


Figure D.1: Node Console Waiting For Disk Encryption Passphrase During Node-installer Run

The passphrase is automatically generated and automatically provided by CMDaemon, and CMDaemon at this stage passes on the message that the node is waiting for the disk encryption passphrase.

To confirm approval of the passphrase, the `installerinteractions` command (section 5.4.4) can be run. This does not run at a disk level, but at a grouping level (node, category, group, chassis, rack ...). Thus, for example, for a node with multiple encrypted disks, it only needs to be confirmed once.

If the `installerinteractions` command is run at node level for a particular node, then it displays information about the installer status and interactions.

The following shows the output at various stages when a node `node001` with the preceding XML configuration is booting up:

Example

```
[basecm11->device[node001]]% installerinteractions
No remote interactions pending.
[basecm11->device[node001]]%
Wed Jun  9 12:32:00 2021 [notice] basecm11: node001 [ INSTALLING ] (node installer started)
[basecm11->device[node001]]%
Wed Jun  9 12:32:11 2021 [notice] basecm11: node001 [ INSTALLING ] (waiting for disk encryption passphrase)
[basecm11->device[node001]]% installerinteractions
Hostname  Category  Type                       Timestamp                  Action               Status
--------- --------- -------------------------- -------------------------- -------------------- --------
node001   default   disk encryption passphrase  Wed Jun  9 12:32:09 2021   Waiting for approval  pending
```

To confirm the approval, the `installerinteractions` command can be run with the `--confirm` option and the `-w|--write` option:

Example

```
[basecm11->device[node001]]% installerinteractions -w --confirm
[basecm11->device[node001]]% installerinteractions
Hostname  Category  Type                       Timestamp                  Action               Status
--------- --------- -------------------------- -------------------------- -------------------- ---------
node001   default   disk encryption passphrase  Wed Jun  9 12:35:22 2021   Waiting for approval  confirmed

[... after some time...]

[basecm11->device[node001]]% installerinteractions
No remote interactions pending.
```

If the `-w|--write` option is not used, then the command is a dry-run, which means it only pretends to write changes.

More options for `installerinteractions` can be seen by running `help installerinteractions`.

Allowing Passphrase Confirmation (To Make An Encrypted Partition Accessible) Until A Future Time

The `installerinteractions` confirmation can be done for a specified time in advance with:

Example

```
[basecm11->device[node001]]% installerinteractions -w --confirm --for 5m
[basecm11->device[node001]]% installerinteractions
Hostname  Category  Type                       Timestamp                       Action         Status
--------- --------- -------------------------- ------------------------------- -------------- ----------
node001   default   disk encryption passphrase  Thu Jun 10 16:53:46 2021        Future action  confirmed
[basecm11->device[node001]]% reboot
```

Possible units that can be used to specify the time period are:

Unit	Meaning
s	seconds
m	minutes
h	hours
d	days
w	weeks

Passphrase Change And Provisioning Mode Considerations

If the passphrase is changed, then the node-installer carries out a `FULL` install (section 5.4.4).

If the confirmation timestamp is set for the future, and

1. if the node passphrase is changed before the time of the timestamp, then the node-installer carries out a `FULL` install too.
2. if the node reboots and reaches the passphrase confirmation state after the time of the timestamp, then the node-installer cannot carry out a `FULL` install too, until a fresh `installerinteractions` confirmation is done.

When the `FULL` install is carried out, the node-installer arranges that the encrypted partitions are provisioned as specified by the XML file, and the node eventually gets to a fully running stage. This happens just as with a regular node, but with the XML `encryption` tags ensuring that the specified partitions are encrypted.

Viewing Disk Partitions And Cipher String Used With The `diskpartitions` Command

Once the node is fully running, information about the partition state can be viewed with the `diskpartitions` command:

Example

```
[basecm11->device[node001]]% diskpartitions
Node     Name     Major    Minor    Blocks      Cipher string    Devices  Device mapper
-------- -------- -------- -------- ----------- ---------------- -------- -------------
...
node001  vdb      253      16       52,428,800
node001  vdb1     253      17       102,400
node001  vdb2     253      18       2,097,152   aes-xts-plain64
node001  vdb3     253      19       50,225,152  aes-xts-plain64
```

LUKS Encryption Defaults

The `<encryption />` tag is quite minimal and implies defaults. Its schema is outlined in section D.1. An expanded example of how LUKS options can be set in the encryption section of the XML file is:

Example

```
<encryption>
    <type>luks2</type>
    <cipher>aes-xts-plain64</cipher>
    <hash>sha256</hash>
    <keySize>256</keySize>
    <name>root</name>
    <allowDiscards />
    <custom>--key-slot=2</custom>
</encryption>
```

D.17.3 Standalone Node: Encrypted Partition XML Example

Nodes are not managed by CMDaemon while in a standalone state. The automated passing of a passphrase by CMDaemon to decrypt the LUKS disk encryption can therefore not take place for standalone nodes. The passphrase must thus be executed directly by the cluster administrator, either at the console, or after re-establishing CMDaemon connectivity.

A new passphrase can be set for the node while it is still managed by CMDaemon, and automatically overwrites the CMDaemon-managed passphrase. The new passphrase can be defined with the `--passphrase` or with the `-p|--payload` <*payload*> option of `installerinteractions`, and needs the `--force` option:

Example

```
Fri Jun 11 23:08:38 2021 [notice] basecm11: node001 [ INSTALLING ] (waiting for disk encryption passphrase)
[basecm11->device[node001]]% installerinteractions -w -p mysecret --confirm --force
Hostname  Category  Type                        Timestamp                  Action                Status
--------- --------- --------------------------- -------------------------- --------------------- ----------
node001   default   disk encryption passphrase  Fri Jun 11 23:08:36 2021   Waiting for approval  pending
[...time passes...]
[basecm11->device[node001]]% installerinteractions
No remote interactions pending.
[basecm11->device[node001]]%
Fri Jun 11 23:09:48 2021 [notice] basecm11: node001 [ INSTALLER_CALLINGINIT ] (switching to local root)
[basecm11->device[node001]]%
Fri Jun 11 23:10:31 2021 [notice] basecm11: node001 [   UP   ]
[basecm11->device[node001]]%
```

The reason for the `--force` option is that changing the passphrase results in the entire disk being reformatted.

Setting a new passphrase means that the node undergoes a `FULL` installation (section 5.4.4) during node provisioning. However for a standalone mode, the provisioning system is bypassed in favor of booting from the local drive, and the passphrase must be entered manually at the node console.

D.17.4 Changing A Passphrase On An Encrypted Node

Passphrases can be of two types:

- `autorandom`: An automatically-generated random passphrase, 256 characters in length
- `custom`: A manually-generated passphrase that is typed in by the cluster administrator. This passphrase is set up when using the

  ```
  installerinteractions --confirm -w
  ```

  command with either the option

  `-p|--payload` <*payload*>
  or the option

  `--passphrase`

If a node with an `autorandom` passphrase type is confirmed with `-p|--payload` or `--passphrase`, then the passphrase type automatically switches to a `custom` passphrase type. The passphrase during this confirmation is set either as the payload (the <*payload*> part) of the `-p|--payload` option, or it is set as a string entered via the prompts that come up when the `--passphrase` option is used.

If echoing of the passphrase to processes viewing utilities is a risk, then it is recommended to use the prompting option, `--passphrase`, instead of the `-p|--payload` option.

Any passphrase change results in a `FULL` installation during node-installer provisioning.

Resetting A Passphrase Type To `autorandom`

To reset a node that is being managed with a `custom` type passphrase back to the `autorandom` type of passphrase, so that the passphrase is managed by CMDaemon management, the `--reset` option of `installerinteractions` can be used:

```
[basecm11->device[node001]]% installerinteractions -w --reset --confirm
```

E

Example initialize And finalize Scripts

The node-installer executes any initialize and finalize scripts at particular stages of its 13-step run during node-provisioning (section 5.4). They are sometimes useful for troubleshooting or workarounds during those stages. The scripts are stored in the CMDaemon database, rather than in the filesystem as plain text files, because they run before the node's init process takes over and establishes the final filesystem.

Default initialize and finalize scripts are provided with the default category:

```
[basecm11->category[default]]% show | grep ize
Initialize script               <1.46KiB>
Finalize script                 <3.4KiB>
```

E.1 When Are They Used?

The initialize and finalize scripts are sometimes used as an alternative configuration option out of a choice of other possible options (section 3.20.1). As a solution it can be a bit of a hack, but sometimes there is no reasonable alternative other than using an initialize or finalize script.

An initialize script: is used well before the init process starts, to execute custom commands before partitions and mounting devices are checked. Typically, initialize script commands are related to partitioning, mounting, or initializing special storage hardware. Often an initialize script is needed because the commands in it cannot be stored persistently anywhere else.

A finalize script: (also run before init, but shortly before init starts) is used to set a file configuration or to initialize special hardware, sometimes after a hardware check. It is run in order to make software or hardware work before, or during the later init stage of boot. Thus, often a finalize script is needed because its commands must be executed before init, and the commands cannot be stored persistently anywhere else, or it is needed because a choice between (otherwise non-persistent) configuration files must be made based on the hardware before init starts.

E.2 Accessing From Base View And cmsh

The initialize and finalize scripts are accessible for viewing and editing:

- In Base View, via the Node Categories or Nodes window, under the Settings window. The navigation paths for these are:
  - Grouping > Node categories[default] > Edit > Settings

  - Devices > Nodes[node001] > Edit > Settings

- In cmsh, using the category or device modes. The get command is used for viewing the script, and the set command to start up the default text editor to edit the script. Output is truncated in the two following examples at the point where the editor starts up:

Example

```
[root@basecm11 ~]# cmsh
[basecm11]% category use default
[basecm11->category[default]]% show | grep script
Parameter                      Value
------------------------------ ------------------------------------------------
Finalize script                <1367 bytes>
Initialize script              <0 bytes>
[basecm11->category[default]]% set initializescript
```

Example

```
[basecm11]% device use node001
[basecm11->device[node001]]%
[basecm11->device[node001]]% set finalizescript
```

E.3 Environment Variables Available To initialize And finalize Scripts

When CMDaemon is fully up, the environment variables available to CMDaemon scripts are fully available and can be listed by the CMDaemon front-ends (page 81).

That full range of environment variables is not available for initialize and finalize scripts, since only a subset of the full environment is defined during the stages associated with the scripts.

For the initialize and finalize scripts, node-specific customizations can still be made from a script using the environment variables that are available. For initialize scripts, this is discussed briefly in section E.5.1.

For finalize scripts, the available environment variables can be listed using the following script as a finalizescript:

Example

```
[basecm11->device[node001]]% get finalizescript
#!/bin/bash
#
# All cluster manager environment variables are prefixed with CMD_
# The root / of the running node is always mounted on /localdisk
#

set | grep CMD_ > /localdisk/var/log/node-installer-finalize.env
```

After the node comes up, the contents of the saved file on that node are the available variables:

Example

```
[root@node001 ~]# cat /var/log/node-installer-finalize.env
CMD_ACTIVE_MASTER_IP=10.141.255.254
CMD_CATEGORY=default
CMD_CHASSIS=
CMD_CHASSIS_IP=0.0.0.0
```

```
CMD_CHASSIS_PASSWORD=
CMD_CHASSIS_SLOT=
CMD_CHASSIS_USERNAME=
...
```

The following table shows the available variables with some example values:

Table E: Environment Variables For The `initialize` *And* `Finalize` *Scripts*

Variable	Example Value
CMD_ACTIVE_MASTER_IP	10.141.255.254
CMD_CATEGORY	default
CMD_CHASSIS	chassis01
CMD_CHASSIS_IP	10.141.1.1
CMD_CHASSIS_PASSWORD	ADMIN
CMD_CHASSIS_SLOT	1
CMD_CHASSIS_USERNAME	ADMIN
CMD_CLUSTERNAME	BCM HEAD Cluster
CMD_DEVICE_HEIGHT	1
CMD_DEVICE_POSITION	10
CMD_DEVICE_TYPE	SlaveNode
CMD_ETHERNETSWITCH	switch01:1
CMD_FSEXPORT__SLASH_cm_SLASH_node-installer_ALLOWWRITE	no
CMD_FSEXPORT__SLASH_cm_SLASH_node-installer_HOSTS	10.141.0.0/16
CMD_FSEXPORT__SLASH_cm_SLASH_node-installer_PATH	/cm/node-installer
CMD_FSEXPORTS	_SLASH_cm_SLASH_node-installer
CMD_FSMOUNT__SLASH_cm_SLASH_shared_DEVICE	master:/cm/shared
CMD_FSMOUNT__SLASH_cm_SLASH_shared_FILESYSTEM	nfs
CMD_FSMOUNT__SLASH_cm_SLASH_shared_MOUNTPOINT	/cm/shared
CMD_FSMOUNT__SLASH_cm_SLASH_shared_OPTIONS	rsize=32768,wsize=32768,\ hard,intr,async
CMD_FSMOUNT__SLASH_dev_SLASH_pts_DEVICE	none
CMD_FSMOUNT__SLASH_dev_SLASH_pts_FILESYSTEM	devpts
CMD_FSMOUNT__SLASH_dev_SLASH_pts_MOUNTPOINT	/dev/pts
CMD_FSMOUNT__SLASH_dev_SLASH_pts_OPTIONS	gid=5,mode=620
CMD_FSMOUNT__SLASH_dev_SLASH_shm_DEVICE	none
CMD_FSMOUNT__SLASH_dev_SLASH_shm_FILESYSTEM	tmpfs
CMD_FSMOUNT__SLASH_dev_SLASH_shm_MOUNTPOINT	/dev/shm
CMD_FSMOUNT__SLASH_dev_SLASH_shm_OPTIONS	defaults
CMD_FSMOUNT__SLASH_home_DEVICE	master:/home
CMD_FSMOUNT__SLASH_home_FILESYSTEM	nfs

...continues

Table E: Environment Variables For The `initialize` *And* `Finalize` *Scripts...continued*

Variable	Example Value
CMD_FSMOUNT__SLASH_home_MOUNTPOINT	home
CMD_FSMOUNT__SLASH_home_OPTIONS	rsize=32768,wsize=32768,\ hard,intr,async
CMD_FSMOUNT__SLASH_proc_DEVICE	none
CMD_FSMOUNT__SLASH_proc_FILESYSTEM	proc
CMD_FSMOUNT__SLASH_proc_MOUNTPOINT	/proc
CMD_FSMOUNT__SLASH_proc_OPTIONS	defaults,nosuid
CMD_FSMOUNT__SLASH_sys_DEVICE	none
CMD_FSMOUNT__SLASH_sys_FILESYSTEM	sysfs
CMD_FSMOUNT__SLASH_sys_MOUNTPOINT	/sys
CMD_FSMOUNT__SLASH_sys_OPTIONS	defaults
CMD_FSMOUNTS *	_SLASH_dev_SLASH_pts _SLASH_proc _SLASH_sys _SLASH_dev_SLASH_shm _SLASH_cm_SLASH_shared _SLASH_home
CMD_GATEWAY	10.141.255.254
CMD_HOSTNAME	node001
CMD_INSTALLMODE	AUTO
CMD_INTERFACE_eth0_IP **	10.141.0.1
CMD_INTERFACE_eth0_MTU **	1500
CMD_INTERFACE_eth0_NETMASK **	255.255.0.0
CMD_INTERFACE_eth0_TYPE **	physical
CMD_INTERFACES *	eth0 eth1 eth2 ipmi0
CMD_IP	10.141.0.1
CMD_MAC	00:00:00:00:00:01
CMD_PARTITION	base
CMD_PASSIVE_MASTER_IP	10.141.255.253
CMD_PDUS	
CMD_POWER_CONTROL	custom
CMD_RACK	rack01
CMD_RACK_HEIGHT	42
CMD_RACK_ROOM	serverroom
CMD_ROLES	sgeclient storage
CMD_SHARED_MASTER_IP	10.141.255.252
CMD_SOFTWAREIMAGE_PATH	/cm/images/default-image
CMD_SOFTWAREIMAGE	default-image
CMD_TAG	00000000a000
CMD_USERDEFINED1	var1
CMD_USERDEFINED2	var2

...continues

Table E: Environment Variables For The `initialize` *And* `Finalize` *Scripts...continued*

Variable	Example Value

* The value for this variable is a string with spaces, not an array. Eg:

CMD_FSMOUNTS="_SLASH_dev_SLASH_pts _SLASH_proc _SLASH_sys _SLASH_dev_SLASH_shm ..."

** The name of this variable varies according to the interfaces available. So,

`eth0` can be replaced by `eth1`, `eth2`, `ipmi0`, and so on.

E.4 Using Environment Variables Stored In Multiple Variables

Some data values, such as those related to interfaces (`CMD_INTERFACES_*`), mount points (`CMD_FSMOUNT__SLASH_*`) and exports (`CMD_FSEXPORT__SLASH_cm__SLASH_node-installer_*`) are stored in multiple variables. The following finalize script set for `node001` shows how they can be used:

Example

```
[head->device*[node001*]]% get finalizescript
#!/bin/bash
echo "These are the interfaces:" >> /localdisk/env
CMD_ENV=`env`
function parser {
  for s in TYPE IP NETMASK; do
    echo $( (grep CMD_INTERFACE_${1/:/-}_${s} | grep -Po "[\w.]+$") <<< "${CMD_ENV[@]}")
  done
}
for interface in $CMD_INTERFACES
do
  read -r type ip mask <<< $(parser $interface)

  echo $interface type=$type >> /localdisk/env
  echo $interface ip=$ip >> /localdisk/env
  echo $interface netmask=$mask >> /localdisk/env
done
```

The technique of storage of values in a file under the path within the node of `/localdisk/` is described later on in section E.5.2. When the node boots up and runs the finalize script, then files stored under `/localdisk/`, end up under the path of `/` after the node is fully up.

The detailed workings of the parser function in the preceding `bash` script are not easy, but the result is that the parser function returns output so that the interface type, IP address, and netmask are listed for each interface. The parser works for physical interfaces, VLAN interfaces, and alias interfaces. For example, if there are two interfaces, `eth0` and `eth0:1`, then the file `env` might be seen to have the following data:

Example

```
[root@head ~]# ssh node001 cat /env
These are the interfaces:
eth0 type=physical
eth0 ip=10.141.0.1
eth0 netmask=255.255.0.0
eth0:1 type=alias
eth0:1 ip=10.141.0.2
eth0:1 netmask=255.255.0.0
```

For remotely mounted devices, the name of the environment variables for mount entries have the following naming convention:

Description	Naming Convention
volume	CMD_FSMOUNT_<x>_DEVICE
mount point	CMD_FSMOUNT_<x>_MOUNTPOINT
filesystem type	CMD_FSMOUNT_<x>_FILESYSTEM
mount point options	CMD_FSMOUNT_<x>_OPTIONS

For the names, the entries <x> are substituted with the local mount point path, such as "/cm/shared", but with the "/" character replaced with the text "_SLASH_". So, for a local mount point path "/cm/shared", the name of the associated volume environment variable becomes CMD_FSMOUNT__SLASH_cm_SLASH_shared_DEVICE.

A similar naming convention is applicable to the names of the environment variables for the export entries:

Description	Naming Convention
exported system writable?	CMD_FSEXPORT_<y>_ALLOWWRITE
allowed hosts or networks	CMD_FSEXPORT_<y>_HOSTS
path on exporter	CMD_FSMOUNT_<y>_PATH

Here, the entry <y> is replaced by the file path to the exported filesystem on the exporting node. This is actually the same as the value of "CMD_FSMOUNT_<y>_PATH", but with the "/" character replaced with the text "_SLASH_".

The entries for the local mount values and the export values in the table in section E.3 are the default values for a newly installed cluster. If the administrator wishes to add more devices and mount entries, this is done by configuring fsexports on the head node, and fsmounts on the regular nodes, using Base View or cmsh (section 3.14).

E.5 Storing A Configuration To A Filesystem

E.5.1 Storing With Initialize Scripts

The initialize script (section 5.4.5) runs after the install-mode type and execution have been determined (section 5.4.4), but before unloading specific drivers and before partitions are checked and filesystems mounted (section 5.4.6). Data output cannot therefore be written to a local drive. It can however be written by the script to the tmpfs, but data placed there is lost quite soon, namely during the pivot_root process that runs when the node-installer hands over control to the init process running from the local drive. However, if needed, the data can be placed on the local drive later by using the finalize script to copy it over from the tmpfs.

Due to this, and other reasons, a finalize script is easier to use for an administrator than an initialize script, and the use of the finalize script is therefore preferred.

E.5.2 Ways Of Writing A Finalize Script To Configure The Destination Nodes

Basic Example—Copying A File To The Image

For a finalize script (section 5.4.11), which runs just before switching from using the ramdrive to using the local hard drive, the local hard drive is mounted under /localdisk. Data can therefore be written to the local hard drive if needed, but is only persistent until a reboot, when it gets rewritten. For example, predetermined configuration files can be written from the NFS drive for a particular node, or they can be written from an image prepared earlier and now running on the node at this stage, overwriting a

node-installer configuration:

Example

```
#!/bin/bash
cp /etc/myapp.conf.overwrite /localdisk/etc/myapp.conf
```

This technique is used in a `finalize` script example in section 3.20.4, except that an append operation is used instead of a copy operation, to overcome a network issue by modifying a network configuration file slightly.

There are three important considerations for most finalize scripts:

1. **Running A Finalize Script Without `exit 0` Considered Harmful**

   Failed finalize script logic flow: For a default configuration without a finalize script, if PXE boot fails from the network during node provisioning, the node then goes on to attempt booting from the local drive via iPXE (section 5.1.2).

   However, if the configuration has a finalize script, such as in the preceding example, and if the finalize script fails, then the failure is passed to the node-installer.

   Avoiding Remote Node Hang During A Finalize Script: `exit 0` Recommended: If the node-installer fails, then no attempt is made to continue booting, and the node remains hung at that stage. This is usually undesirable, and can also make remote debugging of a finalize script annoying.

   Adding an `exit 0` to the end of the finalize script is therefore recommended, and means that an error in the script will still allow the node-installer to continue with an attempt to boot from the local drive.

   Debugging tips when a node hangs during a finalize script: If there is a need to understand the failure, then if the node-installer hangs, the administrator can ssh into the node, and into the node-installer environment, and run the finalize script manually to debug it. Once the bug has been understood, the script can be copied over to the appropriate location in the head node, for nodes or categories.

   Additional aid in understanding a failure may be available by looking through the node-installer logs. The debug mode for the node-installer can be enabled by setting `debug=true` instead of `debug=false` in the file `/cm/node-installer/scripts/node-installer.conf` (for multiarch/multidistro configurations the path takes the form: `/cm/node-installer-`*<distribution>*`-`*<architecture>*`/scripts/node-installer.conf`).

   Another way to help debug a failure could be by setting custom event messages in the script, as explained on page 665.

2. **Protecting a configuration file change from provisioning erasure with `excludelistupdate`:**

   In the preceding example, the finalize script saves a file `/etc/myapp.conf` to the destination nodes.

   To protect such a configuration file from erasure, its file path must be covered in the second sublist in the `excludelistupdate` list (section 5.6.1).

3. **Finalize scripts cannot modify `/proc`, `/sys`, and `/dev` filesystems of end result on node directly:**

   The `/proc`, `/sys`, and `/dev` filesystems are unmounted after the finalize script is run before pivoting into the root filesystem under the `/localdisk` directory, which means any changes made to them

are simply discarded. To change values under these filesystems on the node, an rc.local file inside the software image can be used.

For example, if swappiness is to be set to 20 via the /proc filesystem, one way to do it is to set it in the rc.local file:

Example

```
# cat /cm/images/<image-name>/etc/rc.local | grep -v ^# | grep .
echo 20 > /proc/sys/vm/swappiness
exit 0
# chmod 755 /cm/images/<image-name>/etc/rc.d/rc.local     # must be made executable
```

The preceding way of using rc.local set to run a command to modify the image just for illustration. A better way to get the same result in this case would be to not involve rc.local, but to add a line within the /cm/images/<*image-name*>/etc/sysctl.conf file:

```
vm.swappiness = 20
```

Copying A File To The Image—Decision Based On Detection

Detection within a basic finalize script is useful extra technique. The finalize script example of section 3.20.4 does detection too, to decide if a configuration change is to be done on the node or not.

A further variation on a finalize script with detection is a script selecting from a choice of possible configurations. A symlink is set to one of the possible configurations based on hardware detection or detection of an environment variable. The environment variable can be a node parameter or similar, from the table in section E.3. If it is necessary to overwrite different nodes with different configurations, then the previous finalize script example might become something like:

Example

```
#!/bin/bash
if [[ $CMD_HOSTNAME = node00[1-7] ]]
  then ln -s /etc/myapp.conf.first /localdisk/etc/myapp.conf
fi
if [[ $CMD_HOSTNAME = node01[5-8] ]]
  then ln -s /etc/myapp.conf.second /localdisk/etc/myapp.conf
fi
if [[ $CMD_HOSTNAME = node02[3-6] ]]
  then ln -s /etc/myapp.conf.third /localdisk/etc/myapp.conf
fi
```

In the preceding example, the configuration file in the image has several versions: /etc/myapp.conf.<first|second|third>. Nodes node001 to node007 are configured with the first version, nodes node015 to node018 with the second version, and nodes node023 to node026 with the third version. It is convenient to add more versions to the structure of this decision mechanism.

Copying A File To The Image—With Environment Variables Evaluated In The File

Sometimes there can be a need to use the CMDaemon environment variables within a finalize script to specify a configuration change that depends on the environment.

For example a special service may need a configuration file, test, that requires the hostname myhost, as a parameter=value pair:

Example

```
SPECIALSERVICEPARAMETER=myhost
```

Ideally the placeholder value `myhost` would be the hostname of the node rather than the fixed value `myhost`. Conveniently, the CMDaemon environment variable `CMD_HOSTNAME` has the name of the host as its value.

So, inside the configuration file, after the administrator changes the host name from its placeholder name to the environment variable:

```
SPECIALSERVICE=${CMD_HOSTNAME}
```

then when the node-installer runs the finalize script, the file could be modified in-place by the finalize script, and `${CMD_HOSTNAME}` be substituted by the actual hostname.

A suitable finalize Bash script, which runs an in-line Perl substitution, is the following:

```
#!/bin/bash
perl -p -i -e 's/\$\{([^}]+)\}/defined $ENV{$1} ? $ENV{$1} : $&/eg' /localdisk/some/directory/file
```

Here, `/some/directory/file` means that, if for example the final configuration file path for the node is to be `/var/spool/test` then the file name should be set to `/localdisk/var/spool/test` inside the finalize script.

The finalize script replaces all lines within the file that have environment variable names of the form:

`PARAMETER=${`*<environment variable name>*`}`

with the value of that environment variable. Thus, if *<environment variable name>* is `CMD_HOSTNAME`, then that variable is replaced by the name of the host.

E.5.3 Restricting The Script To Nodes Or Node Categories

As mentioned in section 2.1.3, node settings can be adjusted within a category. So the configuration changes to `ifcfg-eth0` is best implemented per node by accessing and adjusting the `finalize` script per node if only a few nodes in the category are to be set up like this. If all the nodes in a category are to be set up like this, then the changes are best implemented in a `finalize` script accessed and adjusted at the category level. Accessing the scripts at the node and category levels is covered in section E.2.

People used to normal object inheritance behavior should be aware of the following when considering category level and node level finalize scripts:

With objects, a node item value overrules a category level value. On the other hand, finalize scripts, while treated in an analogous way to objects, cannot always inherit properties from each other in the precise way that true objects can. Thus, it is possible that a finalize script run at the node level may not have anything to do with what is changed by running it at the category level. However, to allow it to resemble the inheritance behavior of object properties a bit, the node-level finalize script, if it exists, is always run after the category-level script. This gives it the ability to "overrule" the category level.

F

Workload Managers Quick Reference

F.1 Slurm

Slurm is a GPL-licensed workload management system and developed largely at Lawrence Livermore National Laboratory. The name was originally an acronym for Simple Linux Utility for Resource Management, but the acronym is deprecated because it no longer does justice to the advanced capabilities of Slurm.

The Slurm service and outputs are normally handled using the Base View or `cmsh` front end tools for CMDaemon (section 7.4).

From the command line, direct Slurm commands that may sometimes come in useful include the following:

- `sacct`: used to report job or job step accounting information about active or completed jobs.

  Example

```
# sacct -j 43 -o jobid,AllocCPUs,NCPUS,NNodes,NTasks,ReqCPUs
       JobID  AllocCPUS      NCPUS   NNodes   NTasks  ReqCPUS
------------ ---------- ---------- -------- -------- --------
43                    1          1        1                1
```

- `salloc`: used to allocate resources for a job in real time. Typically this is used to allocate resources and spawn a shell. The shell is then used to execute srun commands to launch parallel tasks.
- `sattach` used to attach standard input, output, and error plus signal capabilities to a currently running job or job step. One can attach to and detach from jobs multiple times.
- `sbatch`: used to submit a job script for later execution. The script typically contains one or more srun commands to launch parallel tasks.
- `sbcast`: used to transfer a file from local disk to local disk on the nodes allocated to a job. This can be used to effectively use diskless compute nodes or provide improved performance relative to a shared filesystem.
- `scancel`: used to cancel a pending or running job or job step. It can also be used to send an arbitrary signal to all processes associated with a running job or job step.
- `scontrol`: the administrative tool used to view and/or modify Slurm state. Note that many scontrol commands can only be executed as user root.

  Example

```
[fred@basecm11 ~]$ scontrol show nodes
NodeName=basecm11 Arch=x86_64 CoresPerSocket=1
   CPUAlloc=0 CPUErr=0 CPUTot=1 CPULoad=0.05 Features=(null)
...
```

If a node, for example `node001`, is stuck in a `CG` state ("completing"), and rebooting it is not feasible, then the following may clear it in some cases:

Example

```
[fred@basecm11 ~]$ scontrol update nodename=node001 state=down reason=hung
[fred@basecm11 ~]$ scontrol update nodename=node001 state=resume
```

- `sinfo`: reports the state of partitions and nodes managed by Slurm. It has a wide variety of filtering, sorting, and formatting options.

  Example

```
basecm11:~ # sinfo -o "%9P %.5a %.10l %.6D %.6t %C %N"
PARTITION AVAIL  TIMELIMIT  NODES  STATE CPUS(A/I/O/T) NODELIST
defq*        up   infinite      1  alloc 1/0/0/1 basecm11
```

- `smap`: reports state information for jobs, partitions, and nodes managed by Slurm, but graphically displays the information to reflect network topology.
- `squeue`: reports the state of jobs or job steps. It has a wide variety of filtering, sorting, and formatting options. By default, it reports the running jobs in priority order and then the pending jobs in priority order.

  Example

```
basecm11:~ # squeue -o "%.18i %.9P %.8j %.8u %.2t %.10M %.6D %C %R"
JOBID PARTITION  NAME  USER ST   TIME  NODES CPUS NODELIST(REASON)
   43      defq  bash  fred  R  16:22      1 1 basecm11
...
```

- `srun`: used to submit a job for execution or initiate job steps in real time. srun has a wide variety of options to specify resource requirements, including: minimum and maximum node count, processor count, specific nodes to use or not use, and specific node characteristics (so much memory, disk space, certain required features, etc.). A job can contain multiple job steps executing sequentially or in parallel on independent or shared nodes within the job's node allocation.

  Auto Scaler dynamic node issues with `srun`: A concern about `srun` (`https://bugs.schedmd.com/show_bug.cgi?id=1333`) at the time of writing (January 2023) is the following: After an `srun` job has been queued, and a new node is added, the `slurm.conf` file is not read again. This means that the new node resource is not seen by jobs using `srun`. Thus, with `srun` jobs, nodes launched dynamically by Auto Scaler remain unused, and can fail. Workarounds are to use `salloc` or `sbatch` instead of `srun`.

- `smap`: reports state information for jobs, partitions, and nodes managed by Slurm, but graphically displays the information to reflect network topology.
- `strigger`: used to set, get or view event triggers. Event triggers include things such as nodes going down or jobs approaching their time limit.

- `sview`: a graphical user interface to get and update state information for jobs, partitions, and nodes managed by Slurm.

There are man pages for these commands. Full documentation on Slurm is available online at: `http://slurm.schedmd.com/documentation.html`.

G

Metrics, Health Checks, Enummetrics, And Actions

This appendix describes the metrics (section G.1), health checks (section G.2), enummetrics (section 10.2.2), and actions (section G.4), along with their parameters, in a newly-installed cluster. Metrics, health checks, enummetrics, and actions can each be standalone scripts, or they can be built-ins. Standalone scripts can be those supplied with the system, or they can be custom scripts built by the administrator. Scripts often require environment variables (as described in section 3.3.1 of the *Developer Manual*). On success scripts must exit with a status of 0, as is the normal practice.

G.1 Metrics And Their Parameters

A list of metric names can be viewed, for example, for the head node, using `cmsh` as follows (section 10.5.3):

```
[basecm11 ~]# cmsh -c "monitoring measurable; list metric"
```

The metrics listed in this section are classed into 10 kinds:

1. regular metrics (section G.1.1)
2. NFS metrics (section G.1.2)
3. InfiniBand metrics (section G.1.3)
4. monitoring system metrics (section G.1.4)
5. GPU metrics (section G.1.6)
6. Job metrics (section G.1.8)
7. IPMI metrics (section G.1.9)
8. Redfish metrics (section G.1.10)
9. SMART metrics (section G.1.11)
10. Prometheus metrics (section G.1.12)

G.1.1 Regular Metrics

Table G.1.1: List Of Metrics

Metric	Description
`AlertLevel`	Indicates the healthiness of a device based on severity of events (section 10.2.8). The lower it is, the better. There are 3 parameters it can take: • `count`: the number of active triggers • `maximum`: the maximum alert level of all active triggers • `sum`: the summed alert level of all active triggers
`BlockedProcesses`	Blocked processes waiting for I/O
`BufferMemory`	System memory used for buffering
`BytesRecv`*,‡	Bytes/s received
`BytesSent`*,‡	Bytes/s sent
`CPUGuest`*	CPU time spent in guest mode (Jiffies/s)
`CPUIdle` *	CPU time spent in idle mode (Jiffies/s)
`CPUIrq` *	CPU time spent in servicing IRQ (Jiffies/s)
`CPUNice` *	CPU time spent in nice mode (Jiffies/s)
`CPUSoftIrq` *	CPU time spent in servicing soft IRQ (Jiffies/s)
`CPUSteal` *	CPU time spent in steal mode (Jiffies/s)
`CPUSystem` *	CPU time spent in system mode (Jiffies/s)
`CPUUser` *	CPU time spent in user mode (Jiffies/s)
`CPUUsage` *	Percent of time not spent in idle mode (sum of non-idling percentages) (%/s)
`CPUWait` *	CPU time spent in I/O wait mode (Jiffies/s).
`CacheMemory`	System memory used for caching.
`Cores`	Number of cores for a node
`CoresDown`	Number of cores for all nodes marked as DOWN
`CoresTotal`	Total number of known cores for all nodes
`CoresUp`	Number of cores for all nodes marked as UP
`CtxtSwitches`*	Context switches/s
`DPUNodesClosed`	Number of DPUs not marked as UP or DOWN
`DPUNodesDown`	Number of DPUs marked as DOWN
`DPUNodesTotal`	Total number of DPUs
`DPUNodesUp`	Number of DPUs not marked as UP or DOWN
`DevicesClosed`	Number of devices not marked as UP or DOWN
`DevicesDown`	Number of devices marked as DOWN
`DevicesTotal`	Total number of devices

...continues

Table G.1.1: List Of Metrics...continued

Metric	Description
DevicesUp	Number of devices in status UP. A node (head, regular, virtual, cloud) or GPU Unit is not classed as a device. A device can be an item such as a switch, PDU, chassis, or rack, if the item is enabled and configured for management.
DropRecv*,‡	Packets/s received and dropped
DropSent*,‡	Packets/s sent and dropped
EC2SpotPrice	Amazon EC2 price for spot instances
EccDBitGPU**	Total number of double bit ECC errors/s (file: sample_gpu)
EccSBitGPU**	Total number of single bit ECC errors/s (file: sample_gpu)
ErrorsRecv*,‡	Packets/s received with error
ErrorsSent*,‡	Packets/s sent with error
FPGAsDown	Number of FPGAs for all nodes marked as DOWN
FPGAsTotal	Total number of known FPGAs for all nodes
FPGAsUp	Number of FPGAs for all nodes marked as UP
FabricTopologies	Number of fabric topologies
FabricTopologyHostUsage	Average usage of all topology hosts
FabricTopologyResourceBoxUsage	Average usage of all topology resource boxes
Forks*	Forked processes/s
FrameErrors*,‡	Packet framing errors/s
FreeFiles§	Free file inodes on the specified mount point
FreeSpace§	Free space for non-root user. Takes mount point as a parameter
GPUUnitsClosed	Number of GPU units not marked as UP or DOWN
GPUUnitsDown	Number of GPU units marked as DOWN
GPUUnitsTotal	Total number of GPU Units
GPUUnitsUp	Number of GPU units marked as UP
GPUsTotal	Total number of known GPUs for all nodes
GPUsUp	Number of GPUs for all nodes marked as UP
GPUsDown	Number of GPUs for all nodes marked as DOWN
HardwareCorruptedMemory	Hardware corrupted memory detected by ECC
IOInProgress†	I/O operations in progress
IOTime*,†	I/O operations time in milliseconds/s
InterfaceState	Interface operation state
IpForwDatagrams*	Input IP datagrams/s to be forwarded/s
IpFragCreates*	IP datagram fragments/s generated/s
IpFragFails*	IP datagrams/s which needed to be fragmented but could not
IpFragOKs*	IP datagrams/s successfully fragmented
IpInAddrErrors*	Input datagrams/s discarded because the IP address in their header was not a valid address
IpInDelivers*	Input IP datagrams/s successfully delivered
IpInDiscards*	Input IP datagrams/s discarded

...continues

Table G.1.1: List Of Metrics...continued

Metric	Description
`IpInHdrErrors`*	Input IP datagrams/s discarded due to errors in their IP headers
`IpInReceives`*	Input IP datagrams/s, including ones with errors, received from all interfaces
`IpInUnknownProtos`*	Input IP datagrams/s received but discarded due to an unknown or unsupported protocol
`IpOutDiscards`*	Output IP datagrams/s discarded
`IpOutNoRoutes`*	Output IP datagrams/s discarded because no route could be found
`IpOutRequests`*	Output IP datagrams/s supplied to IP in requests for transmission
`IpReasmOKs`*	IP datagrams/s successfully re-assembled
`IpReasmReqds`*	IP fragments/s received needing re-assembly
`JobsRunning`	Jobs running on the node
`LiteNodesClosed`	Number of lite nodes not marked as UP or DOWN
`LiteNodesDown`	Number of lite nodes marked as DOWN
`LiteNodesTotal`	Total number of lite nodes
`LiteNodesUp`	Number of lite nodes marked as UP
`LoadFifteen`	Load average on 15 minutes
`LoadFive`	Load average on 5 minutes
`LoadOne`	Load average on 1 minute
`MajorPageFaults`*	Page faults/s that require I/O
`ManagedServicesOk`	This metric uses the `ManagedServicesOk` health check (page 1035) for a grouping of nodes, such as the nodes in a category, or the nodes in a cluster. A parameter is specified for the metric, and a value is returned, as follows: • `fail`: Total number of health checks in a `FAIL` state • `good`: Percentage that are in the `PASS` state • `pass`: Total number that are in the `PASS` state • `total`: Total number of the ManagedServicesOk metrics being sampled, regardless of state • `unknown`: Total number that are in the `UNKNOWN` state

...continues

Table G.1.1: List Of Metrics...continued

Metric	Description
ManagedSwitchesClosed	Number of managed switches not marked as UP or DOWN
ManagedSwitchesDown	Number of managed switches marked as DOWN
ManagedSwitchesTotal	Number of managed switches not marked as UP or DOWN
ManagedSwitchesUp	Total number of managed switches
MemoryAvailable	Available system memory
MemoryFree	Free system memory
MemoryTotal	Total system memory
MemoryUsed[¶]	Used system memory for a process specified as a parameter. Processes can be anything that is always running: **Example** • cm-lite-daemon • cm-mqtt • cmd • mysqld • slurmctld • slurmd
MemoryUtilization	Memory utilization
MergedReads[*,†]	Merged reads/s
MergedWrites[*,†]	Merged writes/s
NodesClosed	Number of nodes not marked as UP or DOWN
NodesDown	Number of nodes marked as DOWN
NodesTotal	Total number of nodes
NodesUp	Number of nodes in status UP
OccupationRate	Cluster occupation rate—a normalized cluster load percentage. 100% means all cores on all nodes are fully loaded. The calculation is done as follows: LoadOne on each node is mapped to a value, calibrated so that LoadOne=1 corresponds to 100% per node. The maximum allowed for a node in the mapping is 100%. The average of these mappings taken over all nodes is the OccupationRate. A high value can indicate the cluster is being used optimally. However, a value that is 100% most of the time suggests the cluster may need to be expanded.
OOM kill	Number of processes killed by OOM killer since boot
PacketsRecv[*,‡]	Packets/s received
PacketsSent[*,‡]	Packets/s sent
PageFaults[*]	Page faults/s
PageIn [*]	Number of bytes the system has paged in from disk/s

...continues

Table G.1.1: List Of Metrics...continued

Metric	Description
PageOut *	Number of bytes the system has paged out to disk/s
PageSwapIn *	Number of bytes the system has swapped in from disk/s
PageSwapOut *	Number of bytes the system has swapped out to disk/s
PDUBankLoad	Total PDU bank load, in amps
PDULoad	Total PDU phase load, in amps
PDUUptime*	PDU uptime per second. I.e. ideally=1, but in practice has jitter effects.
PhaseLoad	Sum of PDULoad over all power distribution units
ProcessCount	Total number of all processes in the OS. These are running processes (RunningProcesses) and blocked processes(BlockedProcesses).
ReadOnly§	Indicates if the specified mount point was mounted as read-only
ReadTime*,†	Read time in milliseconds/s
Reads*,†	Reads/s completed successfully
RunningProcesses	Running processes
ReportedSpeed	Speed reported by the port of the switch. The parameter name used is the hardware port name, as specified by the output of the switchoverview command. For example, for the Cumulus switch these can be swp0, swp1...
SectorsRead*,†	Sectors/s read successfully/s
SectorsWritten*,†	Sectors/s written successfully
SwapCached	Cached swap memory
SwapFree	Free swap memory
SwapTotal	Total swap memory
SwapUsed	Used swap memory
SwapUtilization	Swap memory utilization
SystemTime¶	System time used by process specified by a parameter, for example: cmd
SwitchBroadcastPackets*	Total number of good packets received and directed to the broadcast address/s
SwitchCollisions*	Collisions/s on this network segment
SwitchCPUUsage	Switch CPU utilization estimation (%)
SwitchDelayDiscardFrames*	Frames discarded/s due to excessive transit delay through the bridge
SwitchFilterDiscardFrames*	Valid frames received/s but discarded by the forwarding process
SwitchMTUDiscardFrames*	Number of frames discarded/s due to an excessive size
SwitchMulticastPackets*	Total number of good packets/s received and directed to a multicast address
SwitchOverSizedPackets*	Well-received packets/s longer than 1518 octets
SwitchUnderSizedPackets*	Packets/s received which are less than 64 octets long
SwitchUptime*	Switch uptime per second. Ie, ideally=1, but in practice has jitter effects

...continues

Table G.1.1: List Of Metrics...continued

Metric	Description
TcpCurrEstab	TCP connections that are either ESTABLISHED or CLOSE-WAIT
TcpInErrs*	Input IP segments/s received in error
TcpRetransSegs*	Total number of IP segments/s re-transmitted
ThreadsUsed¶	Threads used by process. For example: cmd
TotalBytesRecv	Total bytes received on swp interfaces over all managed switches
TotalBytesSent	Total bytes sent on swp interfaces over all managed switches
TotalCPUIdle	Cluster-wide core usage in idle tasks (sum of all CPUIdle metric percentages)
TotalCPUPowerUsage	Total CPU power usage over all nodes
TotalCPUSystem	Cluster-wide core usage in system mode (sum of all CPUSystem metric percentages)
TotalCPUTemperature	Average CPU temperature over all nodes
TotalCPUUser	Cluster-wide core usage in user mode (sum of all CPUUser metric percentages)
TotalCPUUtilization	Sum of CPUUsage over all nodes
TotalGPUMemoryUtilization	Average of GPU memory utilization percentage gpu_mem_utilization:average (table G.1.6) over all nodes
TotalGPUNvlinkBandwidth	Total GPU Nvlink bandwidth using gpu_nvlink_total_bandwidth:total (table G.1.6) over all nodes
TotalGPUPowerUsage	Total GPU power usage, using gpu_power_usage:total (table G.1.6) over all nodes
TotalGPUTemperature	Average of average GPU temperature using gpu_temperature:average (table G.1.6) over all nodes
TotalGPUUtilization	Average of GPU utilization using gpu_utilization:average (table G.1.6) over all nodes
TotalMemory	Sum of MemoryTotal over all nodes
TotalMemoryFree	Cluster-wide total of memory free
TotalMemoryUsed	Cluster-wide total of memory used
TotalMemoryUtilization	Sum of MemoryUtilization over all nodes
TotalNodePowerUsage	Total power usage over all nodes
TotalSwap	Sum of SwapTotal over all nodes
TotalSwapFree	Cluster-wide total swap free
TotalSwapUsed	Cluster-wide total swap used
TotalUser	Total number of known users
TotalUserLogin	Total number of logged in users
UdpInDatagrams*	Input UDP datagrams/s delivered to UDP users
UdpInErrors*	Input UDP datagrams/s received that could not be delivered/s for other reasons (no port excl.)
UdpNoPorts*	Received UDP datagrams/s for which there was no application at the destination port

...continues

Table G.1.1: List Of Metrics...continued

Metric	Description
`UniqueUserLogin`	Number of unique users logged in
`Uptime`*	System uptime per second. Ie, ideally=1, but in practice has jitter effects
`UsedFiles`§	Used file inodes on the specified parameter as mount point
`UsedSpace`§	Used space on the specified parameter as mount point
`UserTime`¶	User time used by specified process. For example: `cmd`
`Utilization`¶	Utilization by specified process. For example; `cmd`
`VirtualMemoryUsed`¶	Virtual memory used by specified process. For example `cmd`
`WlmSlotsFree`	The number of WLM slots free
`WlmSlotsTotal`	The total number of WLM slots
`WlmSlotsUsed`	The number of WLM slots in use
`WlmSlotsUtilization`	The percentage of WLM slots in use
`wlm_slurm_state_count`	Number of nodes in the state for a parameter. The possible parameters are: • `allocated` • `completing` • `down` • `drain` • `draining` • `fail` • `failing` • `idle` • `maint` • `mixed`

...continues

Table G.1.1: List Of Metrics...continued

Metric	Description
`WriteTime`*,†	Write time in milliseconds/s (this is a per mille), for the parameter as device
`Writes`*,†	Writes/s completed successfully for the parameter as device
`isilon_node_disk_access_latency`	Isilon access latency for node disk
`isilon_node_disk_iosched_queue`	Isilon iosched queue for node disk
`isilon_node_disk_xfer_size_in`	Transfer size in for node disk
`isilon_node_disk_xfer_size_out`	Transfer size out for node disk
`isilon_node_ip`	IP address of isilon node
`isilon_in_rate`	Bytes written to NFS client from Isilon
`isilon_out_rate`	Bytes read from Isilon to NFS client
`isilon_op_rate`	I/O rate for NFS client to/from Isilon
`node_network_carrier_changes_total`	Samples total network carrier changes
`nvidia_licensed_compute_resources`	Total licensed compute resources
`nvidia_used_compute_resources`	Used compute resources, (maximum of nodes used and GPUs used, but value can only be up to the total licensed compute resources)
`nvidia_used_gpu_resources`	Used GPU resources
`nvidia_used_node_resources`	Used node resources
`nvidia_used_node_resources`	Used node resources

* Cumulative metric. I.e. the metric is derived from cumulative raw measurements taken at two different times, according to: $metric_{time_2} = \frac{(measurement_2 - measurement_1)}{(time_2 - time_1)}$

The metric is a "per second" measurement.

** Standalone scripts, not built-ins.

If sampling from a head node, the script is in directory: `/cm/local/apps/cmd/scripts/metrics/`

For regular nodes, the script is in directory: `/cm/images/default-image/cm/local/apps/cmd/scripts/metrics/`

† Takes block device name (`sda`, `sdc`, `nvme0n1` and so on) as parameter

‡ Takes interface device name (`eth0`, `eth1`, `en01`, `docker0`, `ib0` and so on) as parameter

§ Takes mount point (for example: `/`, or `/var`) as parameter

¶ Takes a process, eg: `cmd` as a parameter

G.1.2 NFS Metrics

The NFS metrics are all cumulative. They correspond to `nfsstat` output, and are shown in table G.1.2.

Table G.1.2: NFS Metrics

NFS Metric	Description
`nfs_client_packet_packets`	NFS client packets statistics: packets
`nfs_client_packet_tcp`	NFS client package statistics: TCP/IP packets
`nfs_client_packet_tcpconn`	NFS client package statistics: TCP/IP connections
`nfs_client_packet_udp`	NFS client package statistics: UDP packets
`nfs_client_rpc_authrefrsh`	NFS Client RPC statistics: authenticated refreshes to RPC server

...continues

Table G.1.2: NFS Metrics

NFS Metric	Description
nfs_client_rpc_calls	NFS Client RPC statistics: calls
nfs_client_rpc_retrans	NFS Client RPC statistics: re-transmissions
nfs_server_file_anon	NFS Server file statistics: anonymous access
nfs_server_file_lookup	NFS Server file statistics: look-ups
nfs_server_file_ncachedir	NFS Server file statistics: ncachedir
nfs_server_file_stale	NFS Server file statistics: stale files
nfs_server_packet_packets	NFS Server packet statistics: packets
nfs_server_packet_tcp	NFS Server packet statistics: TCP/IP packets
nfs_server_packet_tcpconn	NFS Server packet statistics: TCP/IP connections
nfs_server_packet_udp	NFS Server packet statistics: UDP packets
nfs_server_reply_hits	NFS Server reply statistics: hits
nfs_server_reply_misses	NFS Server reply statistics: misses
nfs_server_reply_nocache	NFS Server reply statistics: no cache
nfs_server_rpc_badauth	NFS Server RPC statistics: bad authentication
nfs_server_rpc_badcalls	NFS Server RPC statistics:bad RPC requests
nfs_server_rpc_badclnt	NFS Server RPC statistics: badclnt
nfs_server_rpc_calls	NFS Server RPC statistics: all calls to NFS and NLM
nfs_v3_client_access	NFSv3 client statistics: access
nfs_v3_client_create	NFSv3 client statistics: create
nfs_v3_client_fsinfo	NFSv3 client statistics: static file system information
nfs_v3_client_fsstat	NFSv3 client statistics: dynamic file system status
nfs_v3_client_getattr	NFSv3 client statistics: file system attributes
nfs_v3_client_lookup	NFSv3 client statistics: lookup
nfs_v3_client_pathconf	NFSv3 client statistics: configuration path
nfs_v3_client_read	NFSv3 client statistics: reads
nfs_v3_client_read_readdirplus	NFSv3 client statistics: readdirplus
nfs_v3_client_remove	NFSv3 client statistics: removes
nfs_v3_client_settattr	NFSv3 client statistics: setattr
nfs_v3_client_total	NFSv3 client statistics: total
nfs_v3_client_write	NFSv3 client statistics: writes
nfs_v3_server_access	NFSv3 server statistics: access
nfs_v3_server_create	NFSv3 server statistics: create
nfs_v3_server_fsinfo	NFSv3 server statistics: static file system information
nfs_v3_server_fsstat	NFSv3 server statistics: dynamic file system information
nfs_v3_server_getattr	NFSv3 server statistics: file system attributes gets
nfs_v3_server_lookup	NFSv3 server statistics: file name look-ups
nfs_v3_server_mkdir	NFSv3 server statistics: directory creation
nfs_v3_server_null	NFSv3 server statistics: null operations
nfs_v3_server_pathconf	NFSv3 server statistics: retrieve POSIX information

...continues

Table G.1.2: NFS Metrics

NFS Metric	Description
nfs_v3_server_read	NFSv3 server statistics: reads
nfs_v3_server_readdirplus	NFSv3 server statistics: READDIRPLUS procedures
nfs_v3_server_readlink	NFSv3 server statistics: Symbolic link reads
nfs_v3_server_setattr	NFSv3 server statistics: file system attribute sets
nfs_v3_server_total	NFSv3 server statistics: total
nfs_v3_server_write	NFSv3 server statistics: writes
nfs_v4_server_compound	NFSv4 server statistics: compound operations
nfs_v4_server_null	NFSv4 server statistics: null operations
nfs_v4_server_total	NFSv4 server statistics: total
nfs_v4_servop_access	NFSv4 server statistics: access
nfs_v4_servop_close	NFSv4 server statistics: close
nfs_v4_servop_create	NFSv4 server statistics: create
nfs_v4_servop_getattr	NFSv4 server statistics: file system attributes gets
nfs_v4_servop_getfh	NFSv4 server statistics: filehandle gets
nfs_v4_servop_lookup	NFSv4 server statistics: file name look-ups
nfs_v4_servop_open	NFSv4 server statistics: opens
nfs_v4_servop_putfh	NFSv4 server statistics: filehandle puts
nfs_v4_servop_putrootfh	NFSv4 server statistics: filehandle puts to root
nfs_v4_servop_read	NFSv4 server statistics: reads
nfs_v4_servop_readdirplus	NFSv4 server statistics: READDIRPLUS procedures
nfs_v4_servop_readlink	NFSv4 server statistics: Symbolic link reads
nfs_v4_servop_rename	NFSv4 server statistics: renames
nfs_v4_servop_savefh	NFSv4 server statistics: filehandle saves
nfs_v4_servop_setattr	NFSv4 server statistics: file system attribute sets
nfs_v4_servop_total	NFSv4 server statistics: total
nfs_v4_servop_write	NFSv4 server statistics: writes

G.1.3 InfiniBand Metrics

The available InfiniBand metrics are displayed in table G.1.3.

Table G.1.3: InfiniBand Metrics

InfiniBand Metric	Description
SymbolErrorCount	Total number of minor link errors detected on one or more physical lanes.
LinkErrorRecoveryCount	Total number of times the Port Training state machine has successfully completed the link error recovery process.

...continues

Table G.1.3: InfiniBand Metrics...continued

IB Metric	Description
`LinkDownedCounter`	Total number of times the Port Training state machine has failed the link error recovery process and downed the link.
`PortRcvErrors`	Total number of packets containing an error that were received on the port.
`PortRcvRemotePhysicalErrors`	Total number of packets marked with the EBP delimiter received on the port.
`PortRcvSwitchRelayErrors`	Total number of packets received on the port that were discarded because they could not be forwarded by the switch relay.
`PortXmitDiscards`	Total number of outbound packets discarded by the port because the port is down or congested.
`PortXmitConstrainErrors`	Total number of packets not transmitted from the switch physical port.
`PortRcvConstraintErrors`	Total number of packets received on the switch physical port that are discarded.
`LocalLinkIntegrityErrors`	The number of times that the count of local physical errors exceeded the threshold specified by LocalPhyErrors.
`ExcessiveBufferOverrunError`	The number of times that OverrunErrors consecutive flow control update periods occurred, each having at least one overrun error
`QP1Dropped`	Drops on the lower priority QP1 interconnect
`VL15Dropped`	Drops in the highest priority virtual lane 15
`PortXmitData`	Total number of data octets, divided by 4 (lanes), transmitted on all VLs. This is a 64-bit counter.
`PortRcvData`	Total number of data octets, divided by 4 (lanes), received on all VLs. This is a 64-bit counter.
`PortXmitPkts`	Total number of packets transmitted on all VLs from this port, including packets with errors, and excluding link packets.
`PortRcvPkts`	Total number of packets, including packets containing errors, and excluding link packets, received from all VLs on this port. This is a 64-bit counter.
`PortXmitWait`	The number of ticks during which the port had data to transmit but no data was sent during the entire tick (either because of insufficient credits or because of lack of arbitration).

G.1.4 Monitoring System Metrics

Internal metrics for the monitoring system itself are produced by the `MonitoringSystem` data producer.

The data producer does not run by default on any node, as is seen by running the `nodes` command (page 617) for it:

```
[basecm11->monitoring->setup[MonitoringSystem]]% nodes
Not used
```

However, for convenience, some monitoring system metrics values are displayed on demand in an organized way, by running the `monitoringinfo` command for a device:

Example

```
[basecm11->device[basecm11]]% monitoringinfo
Service                       Queued   Handled    Cache miss Stopped   Suspended  Last operation
----------------------------- -------- ---------- ---------- --------- ---------- ---------------
Mon::CacheGather              4        910        4          no        no         Fri Oct  4 ...
Mon::DataConverter            0        0          0          no        no         -
Mon::DataProcessorEngine      0        155,470    0          no        no         Fri Oct  4 ...
Mon::DataProcessorKeepLatest  0        155,470    0          no        no         Fri Oct  4 ...
Mon::DataProcessorKeepRecent  0        0          0          no        no         -
...
```

The internal monitoring system metrics can be activated so that their data values are saved. This is not recommended, except temporarily for debugging purposes. To save the values, the `Introspect` flag for the monitoring system data producer should be set to `yes`:

Example

```
[basecm11->monitoring->setup[MonitoringSystem]]% set --extra Introspect yes
[basecm11->monitoring->setup*[MonitoringSystem*]]% commit
[basecm11->monitoring->setup[MonitoringSystem]]% nodes
basecm11
[basecm11->monitoring->setup[MonitoringSystem]]% get interval
900
```

After a time defined by `interval`, the metrics can be viewed:

Example

```
[basecm11->device[basecm11]]% latestmetricdata | grep Mon::Storage
Mon::Storage::Engine::elements         Internal/Monitoring/Storage  999,549      44.3s
Mon::Storage::Engine::size             Internal/Monitoring/Storage  1.00 GiB     44.3s
Mon::Storage::Engine::usage            Internal/Monitoring/Storage  9.63%        44.3s
Mon::Storage::Message::elements        Internal/Monitoring/Storage  28           44.3s
...
```

The monitoring system metrics that are saved are shown in table G.1.4.

Table G.1.4: Monitoring System Metrics

Monitoring System Metric	Description
`Mon::CacheGather::handled`	Cache gathers handled/s
`Mon::CacheGather::miss`	Cache gathers missed/s
`Mon::DataProcessor::handled`	Data processors handled/s
`Mon::DataProcessor::miss`	Data processors missed /s
`Mon::DataTranslator::handled`	Data translators handled/s
`Mon::DataTranslator::miss`	Data translators missed/s
`Mon::EntityMeasurableCache::handled`	Measurable cache handled/s
`Mon::EntityMeasurableCache::miss`	Measurable cache missed/s
`Mon::MeasurableBroker::handled`	Measurable broker handled/s
`Mon::MeasurableBroker::miss`	Measurable broker missed/s
`Mon::OOB::TaskService::handled`	Out-of-band task service handled/s
`Mon::OOB::TaskService::miss`	Out-of-band task service missed/s
`Mon::Replicate::Collector::handled`	Replication collection handled/s
`Mon::Replicate::Collector::miss`	Replication collection missed/s
`Mon::Replicate::Combiner::handled`	Replication combiner handled/s
`Mon::Replicate::Combiner::miss`	Replication combiner missed/s
`Mon::RepositoryAllocator::handled`	Repository allocator handled/s
`Mon::RepositoryAllocator::miss`	Repository allocator missed/s
`Mon::RepositoryTrim::handled`	Repository trim handled/s
`Mon::RepositoryTrim::miss`	Repository trim missed/s
`Mon::Storage::Engine::elements`	Storage engine data elements, in total
`Mon::Storage::Engine::size`	Storage engine size, in bytes
`Mon::Storage::Engine::usage`	Storage engine usage
`Mon::Storage::Message::elements`	Storage message data elements, in total
`Mon::Storage::Message::size`	Storage message size in bytes
`Mon::Storage::Message::usage`	Storage message usage
`Mon::Storage::RepositoryId::elements`	Storage repository ID data elements, in total
`Mon::Storage::RepositoryId::size`	Storage repository ID, size, in bytes
`Mon::Storage::RepositoryId::usage`	Repository ID usage
`Mon::TaskInitializer::handled`	Task initializer handled/s
`Mon::TaskInitializer::miss`	Task initializer missed/s
`Mon::TaskSampler::handled`	Task sampler handled/s
`Mon::TaskSampler::miss`	Task sampler missed/s
`Mon::Trigger::Actuator::handled`	Trigger actuators handled/s
`Mon::Trigger::Actuator::miss`	Trigger actuators missed/s
`Mon::Trigger::Dispatcher::handled`	Trigger dispatchers handled/s

...continues

Table G.1.4: Monitoring System Metrics...continued

Monitoring System Metric	Description
Mon::Trigger::Dispatcher::miss	Trigger dispatchers missed/s
Prometheus::DataTranslator::handled	DataTranslator queries handled. The DataTranslator is a translation layer between PromQL format and BCM format sampling.
Prometheus::DataTranslator::miss	DataTranslator misses

G.1.5 CPU Metrics Sampled By The CPUSampler And GPUSampler

Some CPU metrics are sampled by the data producers CPUSampler and GPUSampler (table G.1.5):

Table G.1.5: CPU Metrics Sampled By The CPUSampler And GPUSampler

CPU Metric	Parameter	Description
cpu_core_clock_speed*	cpu_core0	CPU0 core clock speed
cpu_power_cap_enabled*	cpu0	CPU0 power capping enabled
cpu_power_cap_long_term_max_limit*	cpu0	Long term CPU0 power constraint
cpu_power_cap_long_term_time_window*	cpu0	Long term CPU0 power constraint time window
cpu_power_cap_short_term_max_limit*	cpu0	Short term CPU0 power constraint
cpu_power_cap_short_term_time_window*	cpu0	Short term CPU0 power constraint time window
cpu_power_limit*	cpu0	CPU0 power limit
cpu_power_usage*	cpu0	CPU0 power usage
cpu_power_usage	total	Total CPU power usage
cpu_temperature	average	Average CPU temperature
cpu_temperature*	cpu0	CPU0 temperature
cpu_utilization	average	Average CPU utilization

* The number 0 in the parameter column can be replaced by the number associated with the core used. Thus:
cpu_core0 can be replaced by cpu_core1, cup_core2...
and
cpu0 can be replaced by cpu1, cpu2...

G.1.6 GPU Metrics

The data producer (section 10.2.10) for the GPU metrics of this section is GPUSampler.

There were GPU metrics described earlier on in table G.1.1. These were cluster overview metrics about GPUs, and were provided by the ClusterTotal data producer.

However, the NVIDIA GPU metrics of this section, as the GPUSampler data producer name suggests, is about gathering the sampled GPU data from the devices themselves.

There is also a separate section about job GPU metrics (section G.1.8, page 1014), which uses the JobSampler data producer, and is about gathering the sampled job data from the GPU devices themselves.

The device parameter for the GPU metrics in this section, unless otherwise noted, specifies the device slot number that the GPU uses. The parameter takes the form gpu0, gpu1, and so on. It is appended to the metric with a colon character. For example, the gpu_ecc_dbe_agg metric, if used with gpu1, is specified as:

Example

```
gpu_ecc_dbe_agg:gpu1
```

Available GPU metrics for V100 and A100 GPUs are listed in table G.1.6. The available GPU health checks for V100 and A100 GPUs are listed in table G.2.2.

If the cluster has been configured with AMD GPUs (section 7.4 of the *Installation Manual*) then AMD GPU metrics become available. Metrics in the table that are also valid AMD GPU metrics are noted. Some metrics have been added and noted that are only valid for AMD GPUs.

Extra GPU measurables for the GB200 GPUs are listed in section 12.6 of the *NVIDIA Mission Control Manual*.

Table G.1.6: GPU Metrics

GPU Metric	Description
gpu_dec_utilization*	GPU decoding usage
gpu_ecc_dbe_agg	Total double bit aggregate ECC errors
gpu_ecc_dbe_vol	Total double bit volatile ECC errors
gpu_ecc_sbe_agg	Total single bit aggregate ECC errors
gpu_ecc_sbe_vol	Total single bit volatile ECC errors
gpu_enc_utilization*	GPU encoding usage
gpu_enforced_power_limit	GPU-enforced power limit
gpu_mem_clock	GPU memory clock (also for AMD GPU)
gpu_mem_copy_utilization*	Percentage of GPU memory copy used
gpu_mem_free	Amount of GPU free memory
gpu_mem_total**	GPU framebuffer size (also for AMD GPU)
gpu_mem_used**	Amount of GPU memory used (also for AMD GPU)
gpu_mem_utilization*	GPU memory used percentage
gpu_memory_temp*	GPU memory temperature (Celsius)
gpu_nvlink_total_bandwidth**	Total NVLink bandwidth used
gpu_power_management_limit	GPU power management limit
gpu_power_usage**	GPU power usage (also for AMD GPU)
gpu_power_violation**	Throttling duration due to power constraints
gpu_shutdown_temp	GPU shutdown temperature (Celsius)
gpu_slowdown_temp	GPU slowdown temperature (Celsius)
gpu_sm_clock	GPU shader multiprocessor clock (also for AMD GPU)

...continues

Table G.1.6: GPU Metrics...continued

GPU Metric	Description
gpu_temperature*	GPU temperature (Celsius)
gpu_thermal_violation**	Throttling duration due to thermal constraints
gpu_utilization*	Average GPU utilization percentage
gpu_xid_error	The value is the specific XID error

* Specified as average, or specified for a GPU.
For example, for the gpu_dec_utilization metric:
gpu_dec_utilization:average or gpu_dec_utilization:GPU0

** Specified as a total, or specified for a GPU.
For example, for the gpu_mem_total metric:
gpu_mem_total:total or gpu_mem_total:GPU0

G.1.7 GPU Profiling Metrics

The data producer (section 10.2.10) for the GPU profiling metrics of this section is GPUSampler.

The device parameter for the GPU profiling metrics in this section, unless otherwise noted, specifies the device slot number that the GPU uses. The parameter takes the form gpu0, gpu1, and so on. It is appended to the metric with a colon character. For example, the gpu_profiling_fp64_active metric, if used with gpu1, is specified as:

Example

```
gpu_profiling_fp64_active:gpu1
```

Available GPU profiling metrics for V100 and A100 GPUs are listed in table G.1.7.

Table G.1.7: GPU Profiling Metrics

GPU Profiling Metric	Description
gpu_profiling_dram_active	The ratio of cycles the device memory interface is active sending or receiving data
gpu_profiling_fp16_active	Ratio of cycles the fp16 pipe is active
gpu_profiling_fp32_active	Ratio of cycles the fp32 pipe is active
gpu_profiling_fp64_active	Ratio of cycles the fp64 pipe is active
gpu_profiling_graphics_engine_active	Ratio of time the graphics engine is active. The graphics engine is active if a graphics/compute context is bound and the graphics pipe or compute pipe is busy
gpu_profiling_nvlink_read	The number of bytes of active NvLink rx (read) data including both header and payload

...continues

Table G.1.7: GPU Profiling Metrics...continued

GPU Profiling Metric	Description
gpu_profiling_nvlink_transmit	The number of bytes of active NvLink tx (transmit) data including both header and payload
gpu_profiling_pcie_read	The number of bytes of active PCIe rx (read) data including both header and payload
gpu_profiling_pcie_transmit	The number of bytes of active PCIe tx (transmit) data including both header and payload
gpu_profiling_pipe_tensor_active	The ratio of cycles the tensor (HMMA) pipe is active
gpu_profiling_sm_active	The ratio of cycles an SM has at least 1 warp assigned
gpu_profiling_sm_occupancy	The ratio of cycles the tensor (HMMA) pipe is active

G.1.8 Job Metrics

Job metrics are introduced in section 11.1.

Basic Job Metrics

The following table lists some of the most useful job metrics that BCM can monitor and visualize. In the table, the text <*device*> denotes a block device name, such as `sda`.

On virtual machines, block device metrics may be unavailable because of virtualization.

Table G.1.8.1: Basic Job Metrics

Job Metric	Description	Cgroup Source File
`blkio.time:`<*device*>	Time job had I/O access to device	`blkio.time_recursive`
`blkio.sectors:`<*device*>	Sectors transferred to or from specific devices by a cgroup	`blkio.sectors_recursive`
`blkio.io_service_read:`<*device*>	Bytes read	`blkio.io_service_bytes_recursive`
`blkio.io_service_write:`<*device*>	Bytes written	`blkio.io_service_bytes_recursive`
`blkio.io_service_sync:`<*device*>	Bytes transferred synchronously	`blkio.io_service_bytes_recursive`
`blkio.io_service_async:`<*device*>	Bytes transferred asynchronously	`blkio.io_service_bytes_recursive`
`blkio.io_wait_time_read:`<*device*>	Total time spent waiting for service in the scheduler queues for I/O read operations	`blkio.io_wait_time_recursive`
`blkio.io_wait_time_write:`<*device*>	Total time spent waiting for service in the scheduler queues for I/O write operations	`blkio.io_wait_time_recursive`
`blkio.io_wait_time_sync:`<*device*>	Total time spent waiting for service in the scheduler queues for I/O synchronous operations	`blkio.io_wait_time_recursive`
`blkio.io_wait_time_async:`<*device*>	Total time spent waiting for service in the scheduler queues for I/O asynchronous operations	`blkio.io_wait_time_recursive`
`cpuacct.usage`	Total CPU time consumed by all job processes	`cpuacct.usage`
`cpuacct.stat.user`	User CPU time consumed by all job processes	`cpuacct.stat`

...continues

...continued

Job Metric	Description	Cgroup Source File
cpuacct.stat.system	System CPU time consumed by all job processes	cpuacct.stat
memory.usage	Total current memory usage	memory.usage_in_bytes
memory.memsw.usage	Sum of current memory plus swap space usage	memory.memsw.usage_in_bytes
memory.memsw.max_usage	Maximum amount of memory and swap space used	memory.memsw.max_usage_in_bytes
memory.failcnt	How often the memory limit has reached the value set in memory.limit_in_bytes	memory.failcnt
memory.memsw.failcnt	How often the memory plus swap space limit has reached the value set in memory.memsw.limit_in_bytes	memory.memsw.failcnt
memory.swap	Total swap usage	memory
memory.cache	Total page cache, including tmpfs (shmem)	memory
memory.mapped_file	Size of memory-mapped mapped files, including tmpfs (shmem)	memory
memory.unevictable	Memory that cannot be reclaimed	memory

The third column in the table shows the precise source file name that is used when the value is retrieved. These files are all virtual files, and are created as the cgroup controllers are mounted to the cgroup directory. In this case several controllers are mounted to the same directory, which means that all the virtual files will show up in that directory, and in its associated subdirectories—job cgroup directories—when the job runs.

Advanced Job Metrics

The metrics in the preceding table are enabled by default. There are also over 40 other advanced metrics that can be enabled via the `Enable Advanced Metrics` property of the `jobmetricsettings` object:

```
[basecm11->monitoring->setup[JobSampler]->jobmetricsettings]% show
Parameter                      Value
------------------------------ ------------------------------------------------
Enable Advanced Metrics        no
Exclude Devices                loop,sr
Exclude Metrics
Include Devices
Include Metrics
Revision
Sampling Type                  Both
[basecm11->monitoring->setup[JobSampler]->jobmetricsettings]% set enableadvancedmetrics yes
[basecm11->monitoring->setup*[JobSampler*]->jobmetricsettings*]% commit
```

The advanced job metrics are:

Table G.1.8.2: Advanced Job Metrics

Advanced Job Metric	Description	Cgroup Source File
`blkio.io_service_time_read:<device>`	Total time between request dispatch and request completion according to CFQ scheduler for I/O read operations	`blkio.io_service_time_recursive`
`blkio.io_service_time_write:<device>`	Total time between request dispatch and request completion according to CFQ scheduler for I/O write operations	`blkio.io_service_time_recursive`
`blkio.io_service_time_sync:<device>`	Total time between request dispatch and request completion according to CFQ scheduler for I/O synchronous operations	`blkio.io_service_time_recursive`
`blkio.io_service_time_async:<device>`	Total time between request dispatch and request completion according to CFQ scheduler for I/O asynchronous operations	`blkio.io_service_time_recursive`
`blkio.io_serviced_read:<device>`	Read I/O operations	`blkio.io_serviced_recursive`
`blkio.io_serviced_write:<device>`	Write I/O operations	`blkio.io_serviced_recursive`
`blkio.io_serviced_sync:<device>`	Synchronous I/O operations	`blkio.io_serviced_recursive`
`blkio.io_serviced_async:<device>`	Asynchronous I/O operations	`blkio.io_serviced_recursive`
`blkio.io_merged_read:<device>`	Number of block I/Os (requests) merged into requests for I/O read operations	`blkio.io_merged_recursive`
`blkio.io_merged_write:<device>`	Number of block I/Os (requests) merged into requests for I/O write operations	`blkio.io_merged_recursive`
`blkio.io_merged_sync:<device>`	Number of block I/Os (requests) merged into requests for I/O synchronous operations	`blkio.io_merged_recursive`
`blkio.io_merged_async:<device>`	Number of block I/Os (requests) merged into requests for I/O asynchronous operations	`blkio.io_merged_recursive`

...continues

...continued

Advanced Job Metric	Description	Cgroup Source File
`blkio.io_queued_read:<device>`	Number of requests queued for I/O read operations	`blkio.io_queued_recursive`
`blkio.io_queued_write:<device>`	Number of requests queued for I/O write operations	`blkio.io_queued_recursive`
`blkio.io_queued_sync:<device>`	Number of requests queued for I/O synchronous operations	`blkio.io_queued_recursive`
`blkio.io_queued_async:<device>`	Number of requests queued for I/O asynchronous operations	`blkio.io_queued_recursive`
`memory.rss`	Anonymous and swap cache, not including tmpfs (shmem)	`memory`
`memory.pgpgin`	Number of pages paged into memory	`memory`
`memory.pgpgout`	Number of pages paged out of memory	`memory`
`memory.active_anon`	Anonymous and swap cache on active least-recently-used (LRU) list, including tmpfs (shmem)	`memory`
`memory.inactive_anon`	Anonymous and swap cache on inactive LRU list, including tmpfs (shmem)	`memory`
`memory.active_file`	File-backed memory on active LRU list	`memory`
`memory.inactive_file`	File-backed memory on inactive LRU list	`memory`
`memory.hierarchical\_memory_limit`	Memory limit for the Hierarchy that contains the memory cgroup of job	`memory`
`memory.hierarchical\_memsw_limit`	Memory plus swap limit for Hierarchy that contains the memory cgroup of job	`memory`

Job Queue Metrics

Job queue metrics become available after workload manager jobs are run. They are produced by the `JobQueueSampler` data producer.

Table G.1.8: List Of Job Queue Metrics

Metric	Description
AvgJobDuration	Average job duration of current jobs
AvgJobStartDelay	Average job start delay of current jobs
CompletedJobs	Successfully completed jobs
CoresInQueue	Cores in queue
CoresUPInQueue	Active cores in queue
EstimatedDelay	Estimated delay time to execute jobs
FailedJobs	Failed completed jobs
GPUsInQueue	GPUs in queue
GPUsUPInQueue	Active GPUs in queue
JobThroughput	Average number of Jobs finished
NodesInQueue	Number of nodes in the queue
NodesUPInQueue	Active nodes in queue
QueuedJobs	Queued jobs
RunningJobs	Running jobs
RunningJobsMax	Maximal running jobs
RunningJobsUtilization	Running jobs utilization

Job GPU Metrics

The GPU metrics of section G.1.6 have a `GPUSampler` data producer (section 10.2.10). As the name suggests, those metrics are about gathering the sampled GPU data from the devices themselves.

The job GPU metrics of this section have a `JobSampler` data producer. As the name suggests, these metrics are about gathering the sampled job data from the GPU devices themselves.

In other words, the job GPU metrics are essentially the same metrics as in the GPU metrics section, but are valid only for the job. This is reflected in their names, which are identical, except for the `job_` prefix. Other differences of job GPU metrics in comparison with GPU metrics are that

- they are sampled only on the node that hosts the GPUs on which the job ran, and do not take specific GPUs as a parameter.
- the metrics are listed in the `monitoring measurable` mode of `cmsh` only after jobs have run on the GPUs.

Available job GPU metrics for V100 and A100 GPUs are listed in table G.1.8.

If the cluster has been configured with AMD GPUs (section 7.4 of the *Installation Manual*) then AMD job GPU metrics become available. Metrics in the table that are also valid AMD job GPU metrics are noted. Some metrics have been added and noted that are only valid for AMD GPUs.

Table G.1.8: Job GPU Metrics For Node

Job GPU Metric	Description For Job On Node
job_gpu_dec_utilization	GPU decoding usage

...continues

Table G.1.8: Job GPU Metrics For Node...continued

Job GPU Metric	Description For Job On Node
job_gpu_ecc_dbe_agg	Total double bit aggregate ECC errors
job_gpu_ecc_dbe_vol	Total double bit volatile ECC errors
job_gpu_ecc_sbe_agg	Total single bit aggregate ECC errors
job_gpu_ecc_sbe_vol	Total single bit volatile ECC errors
job_gpu_enc_utilization	GPU encoding usage
job_gpu_enforced_power_limit	GPU-enforced power limit
job_gpu_mem_clock	GPU memory clock (also for AMD GPU)
job_gpu_mem_copy_utilization	Percentage of GPU memory copy used
job_gpu_mem_free	Amount of GPU free memory
job_gpu_mem_total	GPU framebuffer size (also for AMD GPU)
job_gpu_mem_used	Amount of GPU memory used (also for AMD GPU)
job_gpu_mem_utilization	GPU memory used percentage
job_gpu_memory_temp	GPU memory temperature
job_gpu_nvlink_total_bandwidth	Total NVLink bandwidth used
job_gpu_power_management_limit	GPU power management limit
job_gpu_power_usage	GPU power usage (also for AMD GPU)
job_gpu_power_violation	Throttling duration due to power constraints
job_gpu_shutdown_temp	GPU shutdown temperature
job_gpu_slowdown_temp	GPU slowdown temperature
job_gpu_sm_clock	GPU shader multiprocessor clock (also for AMD GPU)
job_gpu_temperature	GPU temperature
job_gpu_thermal_violation	Throttling duration due to thermal constraints
job_gpu_utilization	GPU utilization (section 12.4.1)
job_gpu_wasted	GPU wasted (section 12.4.1)
job_gpu_xid_error	The value is the specific XID error

G.1.9 IPMI Metrics

The IPMI metrics correspond to metrics provided by the BMC devices. The metrics available depend on the manufacturer, and are detected by BCM. The metrics listed in the following table are a limited list of what may be detected on a system.

The data producer (section 10.2.10) for the IPMI metrics is the `ipmi` data producer.

Table G.1.9: IPMI Metrics

IPMI Metric	Description
Current_<*number*>**	Current seen by BMC sensor <*number*>, in amps (file: `sample_ipmi`)

...continues

Table G.1.9: IPMI Metrics...continued

IPMI Metric	Description
Exhaust_Temp	Exhaust temperature, in Celsius
FETDRV_PG	PowerEdge voltage sensor on IPMI board
Fan<*number*>_RPM	RPM of Fan<*number*> as seen by BMC (file: sample_ipmi)
Fan_Redundancy	fan redundancy status
Inlet_Temp**	Inlet Temperature, in Celsius (file: sample_ipmi)
M01_VDDQ_PG	PowerEdge CPU voltage sensor M01_VDDQ_PG
M01_VTT_PG	PowerEdge CPU voltage sensor
M23_VDDQ_PG	PowerEdge CPU voltage sensor
M23_VTT_PG	PowerEdge CPU voltage sensor
NDC_PG	PowerEdge system board voltage sensor
PFault_Fail_Safe	PowerEdge sensor
PLL_PG	PowerEdge sensor
PS1_PG_Fail	PowerEdge sensor
PS2_PG_Fail	PowerEdge sensor
Pwr_Consumption	Power consumed by BMC, in watts (file: sample_ipmi)
Temp	Temperature, in Celsius
VSA_PG	PowerEdge voltage sensor
VTT_PG	PowerEdge voltage sensor
Voltage_<*number*>**	Voltage seen by BMC sensor *number*, in Volts (file: sample_ipmi)

** Standalone scripts, not built-ins.
If sampling from a head node, the script is in directory: /cm/local/apps/cmd/scripts/metrics/
For regular nodes, the script is in directory: /cm/images/default-image/cm/local/apps/cmd/scripts/metrics/

G.1.10 Redfish Metrics

By default Redfish metrics are sampled only when the BMC interface (section 3.7) starts with rf (e.g. rf0, rf1 etc). The Redfish metric sampler is enabled by setting userdefinedresources as follows:

Example

```
[basecm11->device[node001]]% set userdefinedresources redfish
[basecm11->device[node001]]% commit
```

Redfish Standalone Script Metrics

An example script that samples Redfish metrics is the standalone script /cm/images/default-image/cm/local/apps/cmd/scripts/metrics/sample_redfish:

Table G.1.10: Redfish Standalone Script Metrics

Redfish Standalone Script Metrics	Description
fan_speed	Speed of fan (RPM)
psu_power	The average power supply unit power (W)
sensor_reading	Temperature sensor reading (C)

Redfish Metrics From CMDaemon Built-in Sampling

CMDaemon built-in sampling also samples Redfish metrics from the hardware:

Example

```
[basecm11->device[node001]]% latestmetricdata | head -2; latestmetricdata | grep ^RF
Measurable                                    Parameter Type                                  Value    ...
--------------------------------------------- --------- ------------------------------------- ---------
RF_Baseboard_0_PCB_0_Temp_0                   reading   Environmental/Redfish/Sensor reading  34 C
RF_Baseboard_0_PCB_1_Temp_0                   reading   Environmental/Redfish/Sensor reading  33.5 C
RF_Baseboard_0_PCB_2_Temp_0                   reading   Environmental/Redfish/Sensor reading  33 C
RF_Baseboard_0_StandbyHSC_0_Power_0           reading   Environmental/Redfish/Sensor reading  34.746 W
RF_Baseboard_0_StandbyHSC_0_Temp_0            reading   Environmental/Redfish/Sensor reading  33.4375 C
RF_C2C_0_Resource_MaxSpeed                              Environmental/Redfish/Port            0 B/s
RF_DIMM_Slot_AllowedSpeeds                              Environmental/Redfish/Memory          no data
RF_DIMM_Slot_Capacity                                   Environmental/Redfish/Memory          0 B
RF_DIMM_Slot_CapacityMiB                                Environmental/Redfish/Memory          0 B
RF_DIMM_Slot_Nvidia_RowRemappingFailed                  Environmental/Redfish/Memory          0
RF_GPU_0_DRAM_0_Memory_Metrics_Bandwidth                Environmental/Redfish/Memorymetrics   0.0%
RF_GPU_0_DRAM_0_Memory_Metrics_CapacityUtilizat+        Environmental/Redfish/Memorymetrics   0.0%
RF_GPU_0_Processor_Metrics_Bandwidth                    Environmental/Redfish/Processormetrics 0.0%
RF_GPU_0_Processor_Metrics_LifeTime_Correctable+        Environmental/Redfish/Processormetrics 0
RF_GPU_0_Processor_Metrics_LifeTime_Uncorrectab+        Environmental/Redfish/Processormetrics 0
RF_GPU_0_Processor_Metrics_Nvidia_AccumulatedGP+        Environmental/Redfish/Processormetrics no data
RF_GPU_0_Processor_Metrics_Nvidia_AccumulatedSM+        Environmental/Redfish/Processormetrics no data
RF_GPU_0_Processor_Metrics_Nvidia_DMMAUtilizatio        Environmental/Redfish/Processormetrics 0.0%
RF_GPU_0_Processor_Metrics_Nvidia_FP16Activity          Environmental/Redfish/Processormetrics 0.0%
```

The preceding list is a truncated excerpt from an NVIDIA DGX system. The Redfish metrics that are available depend on the vendor and hardware.

Further examples of Redfish measurables on an NVIDIA DGX system can be seen in section 12.8 (metrics), section 12.9 (enums), and section 12.10 (health checks) of the *NVIDIA Mission Control Manual*.

G.1.11 SMART Metrics

The SMART metrics correspond to metrics provided by SMART hard drive implementation. The metrics available depend on the manufacturer, and are detected by BCM. The metrics listed in the following table are a limited list of what may be detected on a system.

A SMART metric takes a block device name (`sda`, `sdc`, `nvme0n1` and so on) as a parameter. Some reported values, such as "spin up time" and "start/stop count" are not relevant on non-rotational (solid-state) drives.

The data producer (section 10.2.10) for SMART metrics is the `smart` data producer.

Table G.1.11: SMART Metrics

SMART Metric	Description
`Command_Timeout`	Command timeout
`Current_Pending_Sector`	Current pending sectors
`Hardware_ECC_Recovered`	Hardware ECC recovered
`Offline_Uncorrectable`	Uncorrectable sectors
`Raw_Read_Error_Rate`	Raw read error rate
`Reallocated_Sector_Ct`	Reallocated sectors count
`Reported_Uncorrect`	Reported uncorrectable errors
`UDMA_CRC_Error_Count`	UDMA CRC errors

G.1.12 Prometheus Metrics

Prometheus metrics are introduced in section 12.2.

The data producers for Prometheus metrics are JobSampler, JobMetadataSampler, and some others.

Table G.1.12: Prometheus Metrics

Prometheus Metric	Description (for a job, unless asterisked)
`job_blkio_io_merged`	Number of block I/Os (requests) merged into requests for I/O operations by a cgroup
`job_blkio_io_queued`	Number of requests queued for I/O operations by a cgroup
`job_blkio_io_service_bytes`	Reports the number of bytes transferred to or from specific devices by a cgroup as seen by the CFQ scheduler
`job_blkio_io_service_bytes_total`	Reports the number of bytes transferred to or from specific devices by a cgroup as seen by the CFQ scheduler (for all jobs)*
`job_blkio_io_service_time_seconds`	Reports the total time in seconds between request dispatch and request completion for I/O operations on specific devices by a cgroup as seen by the CFQ scheduler
`job_blkio_io_serviced`	Reports the number of I/O operations performed on specific devices by a cgroup as seen by the CFQ scheduler
`job_blkio_io_wait_time_seconds`	Reports the total time I/O operations on specific devices by a cgroup spent waiting for service in the scheduler queues

...continues

Table G.1.12: Prometheus Metrics...continued

Prometheus Metric	Description (for a job, unless asterisked)
job_blkio_sectors	Reports the number of sectors transferred to or from specific devices by a cgroup
job_blkio_time_seconds	Reports the time that a cgroup had I/O access to specific devices
job_cpuacct_stat_system	System CPU time consumed by processes
job_cpuacct_stat_user	User CPU time consumed by processes
job_cpuacct_usage_seconds	CPU usage time consumed
job_memory_active_anon_bytes	Anonymous and swap cache on active least-recently-used (LRU) list, including tmpfs (shmem), in bytes
job_memory_active_file_bytes	File-backed memory on active LRU list, in bytes
job_memory_cache_bytes	Page cache, including tmpfs (shmem), in bytes
job_memory_failcnt	Reports the number of times that the memory limit has reached the value set in memory.limit_in_bytes
job_memory_hierarchical_memory_limit_bytes	Memory limit for the hierarchy that contains the memory cgroup, in bytes
job_memory_hierarchical_memsw_limit_bytes	Memory plus swap limit for the hierarchy that contains the memory cgroup, in bytes
job_memory_inactive_anon_bytes	Anonymous and swap cache on inactive LRU list, including tmpfs (shmem), in bytes
job_memory_inactive_file_bytes	File-backed memory on inactive LRU list, in bytes
job_memory_mapped_file_bytes	Size of memory-mapped mapped files, including tmpfs (shmem), in bytes
job_memory_memsw_failcnt	Reports the number of times that the memory plus swap space limit has reached the value set in memory.memsw.limit_in_bytes
job_memory_memsw_max_usage_bytes	Reports the maximum amount of memory and swap space used by processes in the cgroup, in bytes
job_memory_memsw_usage_bytes	Reports the sum of current memory usage plus swap space used by processes in the cgroup, in bytes
job_memory_pgpgin_bytes	Number of pages paged into memory
job_memory_pgpgout_bytes	Number of pages paged out of memory

...continues

Table G.1.12: Prometheus Metrics...continued

Prometheus Metric	Description (for a job, unless asterisked)
job_memory_rss_bytes	Anonymous and swap cache, not including tmpfs (shmem), in bytes
job_memory_swap_bytes	Swap usage, in bytes
job_memory_unevictable_bytes	Memory that cannot be reclaimed, in bytes,
job_memory_usage_bytes	Reports the total current memory usage by processes in the cgroup, in bytes
job_memory_usage_bytes_total	Reports the total current memory usage by processes in the cgroup, in bytes (for all jobs)*
job_metadata_allocated_cpu_cores	CPU cores used by a job by the user
job_metadata_allocated_gpus	GPUs used by a job by the user
job_metadata_is_running	Returns 1 if the job metadata sampler is running a job
job_metadata_is_waiting	Returns 1 if the job metadata sampler is waiting
job_metadata_num_cpus	Number of CPUs that the job runs on
job_metadata_num_nodes	Number of nodes that the job runs on
job_metadata_pending_jobs	Number of pending jobs for the user
job_metadata_running_jobs	Number of running jobs for the user
job_metadata_running_seconds	Time the job has run
job_metadata_waiting_seconds	Time the job has been waiting to run
users_job_effective_cpu_seconds:1w	CPU seconds used by users over the past week*
users_job_running_count:1w	Number of jobs run by users over the past week*
users_job_waiting_seconds:1w	Time users have been waiting for jobs to run over the past week*
users_job_wall_clock_seconds:1w	Time the jobs runs for users according to wall time over the past week*
users_job_wasted_cpu_seconds:1w	CPU time wasted during users jobs over the past week*

* total in the past 7x24x60x60 seconds, as measured at the time of sampling

G.1.13 NetQ Metrics

NetQ metrics (table G.1.13) are metrics sourced from NetQ. Configuring NetQ with BCM is described in section 3.11.

The data producer (section 10.2.10) for NetQ metrics is the `netq` data producer.

Table G.1.13: NetQ Metrics

NetQ Metric	Description
NetQ_node_fan_speed	Fan speed (RPM)

...continues

Table G.1.13: NetQ Metrics...continued

NetQ Metric	Description
NetQ_node_nvlink_rx_all_flits	NVLink packets received, control and data (FLITs/s)
NetQ_node_nvlink_rx_data_flits	NVLink packets received, data (FLITs/s)
NetQ_node_nvlink_tx_all_flits	NVLink packets sent, control and data (FLITs/s)
NetQ_node_nvlink_tx_data_flits	NVLink packets sent, data (FLITs/s)
NetQ_node_PSU_power_input	Power supply power input (W)
NetQ_node_PSU_power_output	Power supply power output (W)
NetQ_node_PSU_voltage_input	Power supply voltage input (V)
NetQ_node_PSU_voltage_output	Power supply voltage output (V)
NetQ_node_Temp_sensor	Temperature sensor value (C)

NetQ health checks are covered in section G.2.4.

G.1.14 Kubernetes Metrics

Configuring Kubernetes with BCM is described in Chapter 4 of the *Containerization Manual*.

There were cluster Kubernetes metrics described earlier on in table G.1.1. Those were overview metrics about Kubernetes.

The Kubernetes metrics in the following table G.1.14 are state metrics sourced from Kubernetes itself. The data producer (section 10.2.10) for the Kubernetes metrics of this section is `kubestatemetrics`.

The metrics that are not tagged as [STABLE] are experimental, and may change in behavior during Kubernetes updates.

Table G.1.14: Kubernetes Metrics

Kubernetes Metric	Description
kube_configmap_annotations	Kubernetes annotations converted to Prometheus labels.
kube_configmap_created	[STABLE] Unix creation timestamp
kube_configmap_info	[STABLE] Information about configmap.
kube_configmap_labels	[STABLE] Kubernetes labels converted to Prometheus labels.
kube_configmap_metadata_resource_version	Resource version representing a specific version of the configmap.
kube_cronjob_annotations	Kubernetes annotations converted to Prometheus labels.
kube_cronjob_created	[STABLE] Unix creation timestamp
kube_cronjob_info	[STABLE] Info about cronjob.

...continues

Table G.1.14: Kubernetes Metrics...continued

Kubernetes Metric	Description
kube_cronjob_labels	[STABLE] Kubernetes labels converted to Prometheus labels.
kube_cronjob_metadata_resource_version	[STABLE] Resource version representing a specific version of the cronjob.
kube_cronjob_next_schedule_time	[STABLE] Next time the cronjob should be scheduled. The time after lastScheduleTime, or after the cron job's creation time if it's never been scheduled. Use this to determine if the job is delayed.
kube_cronjob_spec_failed_job_history_limit	Failed job history limit tells the controller how many failed jobs should be preserved.
kube_cronjob_spec_successful_job_history_limit	Successful job history limit tells the controller how many completed jobs should be preserved.
kube_cronjob_spec_suspend	[STABLE] Suspend flag tells the controller to suspend subsequent executions.
kube_cronjob_status_active	[STABLE] Active holds pointers to currently running jobs.
kube_cronjob_status_last_schedule_time	[STABLE] LastScheduleTime keeps information of when was the last time the job was successfully scheduled.
kube_cronjob_status_last_successful_time	LastSuccessfulTime keeps information of when was the last time the job was completed successfully.
kube_daemonset_annotations	Kubernetes annotations converted to Prometheus labels.
kube_daemonset_created	[STABLE] Unix creation timestamp
kube_daemonset_labels	[STABLE] Kubernetes labels converted to Prometheus labels.
kube_daemonset_metadata_generation	[STABLE] Sequence number representing a specific generation of the desired state.
kube_daemonset_status_current_number_scheduled	[STABLE] The number of nodes running at least one daemon pod and are supposed to.
kube_daemonset_status_desired_number_scheduled	[STABLE] The number of nodes that should be running the daemon pod.
kube_daemonset_status_number_available	[STABLE] The number of nodes that should be running the daemon pod and have one or more of the daemon pod running and available

...continues

Table G.1.14: Kubernetes Metrics...continued

Kubernetes Metric	Description
kube_daemonset_status_number_misscheduled	[STABLE] The number of nodes running a daemon pod but are not supposed to.
kube_daemonset_status_number_ready	[STABLE] The number of nodes that should be running the daemon pod and have one or more of the daemon pod running and ready.
kube_daemonset_status_number_unavailable	[STABLE] The number of nodes that should be running the daemon pod and have none of the daemon pod running and available
kube_daemonset_status_observed_generation	[STABLE] The most recent generation observed by the daemon set controller.
kube_daemonset_status_updated_number_scheduled	[STABLE] The total number of nodes that are running updated daemon pod
kube_deployment_annotations	Kubernetes annotations converted to Prometheus labels.
kube_deployment_created	[STABLE] Unix creation timestamp
kube_deployment_labels	[STABLE] Kubernetes labels converted to Prometheus labels.
kube_deployment_metadata_generation	[STABLE] Sequence number representing a specific generation of the desired state.
kube_deployment_spec_paused	[STABLE] Whether the deployment is paused and will not be processed by the deployment controller.
kube_deployment_spec_replicas	[STABLE] Number of desired pods for a deployment.
kube_deployment_spec_strategy_rollingupdate_max_surge	[STABLE] Maximum number of replicas that can be scheduled above the desired number of replicas during a rolling update of a deployment.
kube_deployment_spec_strategy_rollingupdate_max_unavailable	[STABLE] Maximum number of unavailable replicas during a rolling update of a deployment.
kube_deployment_status_condition	[STABLE] The current status conditions of a deployment.
kube_deployment_status_observed_generation	[STABLE] The generation observed by the deployment controller.
kube_deployment_status_replicas	[STABLE] The number of replicas per deployment.
kube_deployment_status_replicas_available	[STABLE] The number of available replicas per deployment.

...continues

Table G.1.14: Kubernetes Metrics...continued

Kubernetes Metric	Description
kube_deployment_status_replicas_ready	[STABLE] The number of ready replicas per deployment.
kube_deployment_status_replicas_unavailable	[STABLE] The number of unavailable replicas per deployment.
kube_deployment_status_replicas_updated	[STABLE] The number of updated replicas per deployment.
kube_endpoint_address	[STABLE] Information about Endpoint available and non available addresses.
kube_endpoint_address_available	(Deprecated since v2.6.0) Number of addresses available in endpoint.
kube_endpoint_address_not_ready	(Deprecated since v2.6.0) Number of addresses not ready in endpoint.
kube_endpoint_annotations	Kubernetes annotations converted to Prometheus labels.
kube_endpoint_created	[STABLE] Unix creation timestamp
kube_endpoint_info	[STABLE] Information about endpoint.
kube_endpoint_labels	[STABLE] Kubernetes labels converted to Prometheus labels.
kube_endpoint_ports	[STABLE] Information about the Endpoint ports.
kube_ingress_annotations	Kubernetes annotations converted to Prometheus labels.
kube_ingress_created	[STABLE] Unix creation timestamp
kube_ingress_info	[STABLE] Information about ingress.
kube_ingress_labels	[STABLE] Kubernetes labels converted to Prometheus labels.
kube_ingress_metadata_resource_version	Resource version representing a specific version of ingress.
kube_ingress_path	[STABLE] Ingress host, paths and backend service information.
kube_job_annotations	Kubernetes annotations converted to Prometheus labels.
kube_job_complete	[STABLE] The job has completed its execution.
kube_job_created	[STABLE] Unix creation timestamp
kube_job_info	[STABLE] Information about job.
kube_job_labels	[STABLE] Kubernetes labels converted to Prometheus labels.

...continues

Table G.1.14: Kubernetes Metrics...continued

Kubernetes Metric	Description
kube_job_owner	[STABLE] Information about the Job's owner.
kube_job_spec_completions	[STABLE] The desired number of successfully finished pods the job should be run with.
kube_job_spec_parallelism	[STABLE] The maximum desired number of pods the job should run at any given time.
kube_job_status_active	[STABLE] The number of actively running pods.
kube_job_status_completion_time	[STABLE] CompletionTime represents time when the job was completed.
kube_job_status_failed	[STABLE] The number of pods which reached Phase Failed and the reason for failure.
kube_job_status_start_time	[STABLE] StartTime represents time when the job was acknowledged by the Job Manager.
kube_job_status_succeeded	[STABLE] The number of pods which reached Phase Succeeded.
kube_lease_owner	Information about the Lease's owner.
kube_lease_renew_time	Kube lease renew time.
kube_mutatingwebhookconfiguration_created	Unix creation timestamp.
kube_mutatingwebhookconfiguration_info	Information about the MutatingWebhookConfiguration.
kube_mutatingwebhookconfiguration_metadata_resource_version	Resource version representing a specific version of the MutatingWebhookConfiguration.
kube_mutatingwebhookconfiguration_webhook_clientconfig_service	Service used by the apiserver to connect to a mutating webhook.
kube_namespace_annotations	Kubernetes annotations converted to Prometheus labels.
kube_namespace_created	[STABLE] Unix creation timestamp
kube_namespace_labels	[STABLE] Kubernetes labels converted to Prometheus labels.
kube_namespace_status_condition	The condition of a namespace.
kube_namespace_status_phase	[STABLE] kubernetes namespace status phase.
kube_node_annotations	Kubernetes annotations converted to Prometheus labels.
kube_node_created	[STABLE] Unix creation timestamp
kube_node_info	[STABLE] Information about a cluster node.
kube_node_labels	[STABLE] Kubernetes labels converted to Prometheus labels.

...continues

Table G.1.14: Kubernetes Metrics...continued

Kubernetes Metric	Description
kube_node_role	The role of a cluster node.
kube_node_spec_taint	[STABLE] The taint of a cluster node.
kube_node_spec_unschedulable	[STABLE] Whether a node can schedule new pods.
kube_node_status_allocatable	[STABLE] The allocatable for different resources of a node that are available for scheduling.
kube_node_status_capacity	[STABLE] The capacity for different resources of a node.
kube_node_status_condition	[STABLE] The condition of a cluster node.
kube_pod_completion_time	[STABLE] Completion time in unix timestamp for a pod.
kube_pod_container_info	[STABLE] Information about a container in a pod.
kube_pod_container_resource_limits	The number of requested limit resource by a container. It is recommended to use the kube_pod_resource_limits metric exposed by kube-scheduler instead, as it is more precise.
kube_pod_container_resource_requests	The number of requested request resource by a container. It is recommended to use the kube_pod_resource_requests metric exposed by kube-scheduler instead, as it is more precise.
kube_pod_container_state_started	[STABLE] Start time in unix timestamp for a pod container.
kube_pod_container_status_last_terminated_exitcode	Describes the exit code for the last container in terminated state.
kube_pod_container_status_last_terminated_reason	Describes the last reason the container was in terminated state.
kube_pod_container_status_ready	[STABLE] Describes whether the containers readiness check succeeded.
kube_pod_container_status_restarts_total	[STABLE] The number of container restarts per container.
kube_pod_container_status_running	[STABLE] Describes whether the container is currently in running state.
kube_pod_container_status_terminated	[STABLE] Describes whether the container is currently in terminated state.
kube_pod_container_status_terminated_reason	Describes the reason the container is currently in terminated state.

...continues

Table G.1.14: Kubernetes Metrics...continued

Kubernetes Metric	Description
kube_pod_container_status_waiting	[STABLE] Describes whether the container is currently in waiting state.
kube_pod_container_status_waiting_reason	[STABLE] Describes the reason the container is currently in waiting state.
kube_pod_created	[STABLE] Unix creation timestamp
kube_pod_deletion_timestamp	Unix deletion timestamp
kube_pod_info	[STABLE] Information about pod.
kube_pod_init_container_info	[STABLE] Information about an init container in a pod.
kube_pod_init_container_resource_limits	The number of requested limit resource by an init container.
kube_pod_init_container_resource_requests	The number of requested request resource by an init container.
kube_pod_init_container_status_last_terminated_reason	Describes the last reason the init container was in terminated state.
kube_pod_init_container_status_ready	[STABLE] Describes whether the init containers readiness check succeeded.
kube_pod_init_container_status_restarts_total	[STABLE] The number of restarts for the init container.
kube_pod_init_container_status_running	[STABLE] Describes whether the init container is currently in running state.
kube_pod_init_container_status_terminated	[STABLE] Describes whether the init container is currently in terminated state.
kube_pod_init_container_status_terminated_reason	Describes the reason the init container is currently in terminated state.
kube_pod_init_container_status_waiting	[STABLE] Describes whether the init container is currently in waiting state.
kube_pod_init_container_status_waiting_reason	Describes the reason the init container is currently in waiting state.
kube_pod_ips	Pod IP addresses
kube_pod_owner	[STABLE] Information about the Pod's owner.
kube_pod_restart_policy	[STABLE] Describes the restart policy in use by this pod.
kube_pod_service_account	The service account for a pod.
kube_pod_start_time	[STABLE] Start time in unix timestamp for a pod.
kube_pod_status_container_ready_time	Readiness achieved time in unix timestamp for a pod containers.

...continues

Table G.1.14: Kubernetes Metrics...continued

Kubernetes Metric	Description
kube_pod_status_initialized_time	Initialized time in unix timestamp for a pod.
kube_pod_status_phase	[STABLE] The pods current phase.
kube_pod_status_qos_class	The pods current qosClass.
kube_pod_status_ready	[STABLE] Describes whether the pod is ready to serve requests.
kube_pod_status_ready_time	Readiness achieved time in unix timestamp for a pod.
kube_pod_status_reason	The pod status reasons
kube_pod_status_scheduled	[STABLE] Describes the status of the scheduling process for the pod.
kube_pod_status_scheduled_time	[STABLE] Unix timestamp when pod moved into scheduled status
kube_pod_tolerations	Information about the pod tolerations
kube_poddisruptionbudget_annotations	Kubernetes annotations converted to Prometheus labels.
kube_poddisruptionbudget_created	[STABLE] Unix creation timestamp
kube_poddisruptionbudget_labels	Kubernetes labels converted to Prometheus labels.
kube_poddisruptionbudget_status_current_healthy	[STABLE] Current number of healthy pods
kube_poddisruptionbudget_status_desired_healthy	[STABLE] Minimum desired number of healthy pods
kube_poddisruptionbudget_status_expected_pods	[STABLE] Total number of pods counted by this disruption budget
kube_poddisruptionbudget_status_observed_generation	[STABLE] Most recent generation observed when updating this PDB status
kube_poddisruptionbudget_status_pod_disruptions_allowed	[STABLE] Number of pod disruptions that are currently allowed
kube_replicaset_created	[STABLE] Unix creation timestamp
kube_replicaset_metadata_generation	[STABLE] Sequence number representing a specific generation of the desired state.
kube_replicaset_owner	[STABLE] Information about the ReplicaSet's owner.
kube_replicaset_spec_replicas	[STABLE] Number of desired pods for a ReplicaSet.

...continues

Table G.1.14: Kubernetes Metrics...continued

Kubernetes Metric	Description
kube_replicaset_status_fully_labeled_replicas	[STABLE] The number of fully labeled replicas per ReplicaSet.
kube_replicaset_status_observed_generation	[STABLE] The generation observed by the ReplicaSet controller.
kube_replicaset_status_ready_replicas	[STABLE] The number of ready replicas per ReplicaSet.
kube_replicaset_status_replicas	[STABLE] The number of replicas per ReplicaSet.
kube_replicationcontroller_created	[STABLE] Unix creation timestamp
kube_replicationcontroller_metadata_generation	[STABLE] Sequence number representing a specific generation of the desired state.
kube_replicationcontroller_owner	Information about the ReplicationController's owner.
kube_replicationcontroller_spec_replicas	[STABLE] Number of desired pods for a ReplicationController.
kube_replicationcontroller_status_available_replicas	[STABLE] The number of available replicas per ReplicationController.
kube_replicationcontroller_status_fully_labeled_replicas	[STABLE] The number of fully labeled replicas per ReplicationController.
kube_replicationcontroller_status_observed_generation	[STABLE] The generation observed by the ReplicationController controller.
kube_replicationcontroller_status_ready_replicas	[STABLE] The number of ready replicas per ReplicationController.
kube_replicationcontroller_status_replicas	[STABLE] The number of replicas per ReplicationController.
kube_resourcequota	[STABLE] Information about resource quota.
kube_resourcequota_created	[STABLE] Unix creation timestamp
kube_secret_created	[STABLE] Unix creation timestamp
kube_secret_info	[STABLE] Information about secret.
kube_secret_metadata_resource_version	Resource version representing a specific version of secret.
kube_secret_type	[STABLE] Type about secret.
kube_service_created	[STABLE] Unix creation timestamp
kube_service_info	[STABLE] Information about service.
kube_service_spec_type	[STABLE] Type about service.
kube_statefulset_created	[STABLE] Unix creation timestamp

...continues

Table G.1.14: Kubernetes Metrics...continued

Kubernetes Metric	Description
kube_statefulset_metadata_generation	[STABLE] Sequence number representing a specific generation of the desired state for the StatefulSet.
kube_statefulset_persistentvolumeclaim_retention_policy	Count of retention policy for StatefulSet template PVCs
kube_statefulset_replicas	[STABLE] Number of desired pods for a StatefulSet.
kube_statefulset_status_current_revision	[STABLE] Indicates the version of the StatefulSet used to generate Pods in the sequence [0,currentReplicas).
kube_statefulset_status_observed_generation	[STABLE] The generation observed by the StatefulSet controller.
kube_statefulset_status_replicas	[STABLE] The number of replicas per StatefulSet.
kube_statefulset_status_replicas_available	The number of available replicas per StatefulSet.
kube_statefulset_status_replicas_current	[STABLE] The number of current replicas per StatefulSet.
kube_statefulset_status_replicas_ready	[STABLE] The number of ready replicas per StatefulSet.
kube_statefulset_status_replicas_updated	[STABLE] The number of updated replicas per StatefulSet.
kube_statefulset_status_update_revision	[STABLE] Indicates the version of the StatefulSet used to generate Pods in the sequence [replicas-updatedReplicas,replicas)
kube_storageclass_created	[STABLE] Unix creation timestamp
kube_storageclass_info	[STABLE] Information about storageclass.
kube_validatingwebhookconfiguration_created	Unix creation timestamp.
kube_validatingwebhookconfiguration_info	Information about the ValidatingWebhookConfiguration.
kube_validatingwebhookconfiguration_metadata_resource_version	Resource version representing a specific version of the ValidatingWebhookConfiguration.
kube_validatingwebhookconfiguration_webhook_clientconfig_service	Service used by the apiserver to connect to a validating webhook.

G.1.15 Parameters For Metrics

Metrics have the parameters indicated by the left column in the following example:

Example

```
[basecm11->monitoring->measurable[CPUUser]]% show
Parameter                Value
------------------------ ----------------------------
Class                    CPU
Consolidator             default (ProcStat)
Cumulative               yes
Description              CPU time spent in user mode
Disabled                 no (ProcStat)
Gap                      0 (ProcStat)
Maximal age              0s (ProcStat)
Maximal samples          4,096 (ProcStat)
Maximum                  0
Minimum                  0
Name                     CPUUser
Parameter
Producer                 ProcStat
Revision
Type                     Metric
Unit                     Jiffies/s
```

If the value is inherited from the producer, then it is shown in parentheses next to the value. An inherited value can be overwritten by setting it directly for the parameter of a measurable.

The meanings of the parameters are:

`Class`: A choice assigned to a metric. It can be an internal type, or it can be a standalone class type. A slash (/) is used to separate class levels. A partial list of the class values is:

- `CPU`: CPU-related
- `Disk`: Disk-related
- `Disk/Smart`: SMART Disk-related
- `Fabric`: Fabric-related
- `GPU`: GPU-related
- `Internal`: An internal metric
- `Job`: Job metric
- `License`: License-related
- `Memory`: Memory-related
- `Network`: Network-related
- `OS`: Operating-system-related
- `Process`: Process-related
- `Prometheus`: Prometheus-related
- `Total`: Total cluster-wide-related
- `Workload`: Workload-related
- `Environmental`: Environmental-related

`Consolidator`: This is described in detail in sections 10.4.3 and 10.5.2

`Cumulative`: If set to `no`, then the raw value is treated as not cumulative (for example, `CoresUp`), and the raw value is presented as the metric value.

If set to `yes`, then the metric is treated as being cumulative, which means that a rate (per second) value is presented.

More explicitly: When set to `yes`, it means that the raw sample used to calculate the metric is expected to be cumulative, like, for example, the bytes-received counter for an Ethernet interface. This in turn means that the metric is calculated from the raw value by taking the difference in raw sample measurement values, and dividing it by the time period over which the raw values are sampled. Thus, for example:

- The bytes-received raw measurements, which accumulate as the packets are received, and are in bytes, and have `Cumulative` set to yes, and then have a corresponding metric, `BytesRecv`, with a value in bytes/second.
- The system uptime raw measurements, which accumulate at the rate of 1 second per second, and are in seconds, have `Cumulative` set to yes, and have a corresponding metric, `Uptime`, with a value that uses no units. Ideally, the metric has a value of 1, but in practice the measured value varies a little due to jitter.

`Description`: Description of the raw measurement used by the metric. Empty by default.

`Disabled`: If set to `no` (default) then the metric runs.

`Gap`: The number of samples that are allowed to be missed before a value of `NaN` is set for the value of the metric.

`Maximal age`: the maximum age of RLE samples that are kept. If `Maximal age` is set to `0` then the sample age is not considered. Units can be `w`, `d`, `h`, `m`, `s` (weeks, days, hours, minutes, seconds), with `s` as the default.

`Maximal samples`: the maximum number of RLE samples that are kept. If `Maximal samples` is set to `0` then the number of samples is not considered.

`Maximum`: the value that the y-axis maximum takes in graphs plotted in Base View by default. If the maximum of the y-values is more than the default y-axis maximum value, then the maximum of the y-values becomes the y-axis maximum.

`Minimum`: the value that the y-axis minimum takes in graphs plotted in Base View by default. If the minimum of the y-values is less than the default y-axis minimum value, then the minimum of the y-values becomes the y-axis minimum.

NOTE: To clarify the `Maximum` and `Minimum` concepts, a case can be considered where `minimum`=0, `maximum`=3 are set. If a data point with a y-value of 2 is plotted on a graph, then the y-axis spans the range from 0 to 3 by default. However

- if the data point has a y-value of 4 instead, then it means the y-axis maximum of 3 is re-sized from its default of 3 to the value of 4, so that the y-axis now spans from 0 to 4.
- if the data point has a y-value of -1 instead, then it means the y-axis minimum of 0 is re-sized from its default of 0 to the value of -1, so that the y-axis now spans from -1 to 3.

`Name`: The name given to the metric.

`Parameter`: Parameter used for this metric. For example, `eth0` with the metric `BytesRecv`

`Producer`: The data producer that produces the metric

`Revision`: User-definable revision number for the object

`Type`: This can be one of `metric`, `healthcheck`, or `enummetric`

`Unit`: A unit for the metric. For example: B/s (bytes/second) for `BytesRecv` metric, or unit-less for the `Uptime` metric. A percent is indicated with %

G.2 Health Checks And Their Parameters

A list of health checks can be viewed, for example, for the head node, using `cmsh` as follows (section 10.5.3):

```
[basecm11 ~]# cmsh -c "monitoring measurable; list healthcheck"
```

The health checks listed in this section are classed into 4 kinds:

1. Regular health checks (section 10.2.4) are listed and described in section G.2.1.
2. GPU health checks (section G.2.2)
3. Redfish health checks (section G.2.3)
4. NetQ health checks (section G.2.4)

G.2.1 Regular Health Checks

Table G.2.1: List Of Health Checks

Name	**Query** (script response is `PASS`/`FAIL`)
`ManagedServicesOk`*	Are CMDaemon-monitored services all OK? If the response is `FAIL`, then at least one of the services being monitored is failing. The `latesthealthdata -v` command (section 10.6.3) should show which one(s). After correcting the problem with the service, a reset of the service is normally carried out (section 3.15, page 187). There is also a related `ManagedServicesOk` metric on page 994.
`Mon::Storage`	Is space available for the monitoring system metrics (section G.1.4)?
`chrootprocess`	Are there daemon processes running using chroot in software images? Here: yes = `FAIL`. On failure, kill cron daemon processes running in the software images.
`cm-chroot-sw-img`	Are there dangling mounts left behind of images created by cm-chroot-sw-img? Here: yes = `FAIL`. The cluster administrator is expected to inspect and unmount the dangling mounts.
`cmha-status`	Are both head nodes up and running?
`cmsh` *	Is `cmsh` available?
`cuda-dcgm`	Is `cuda-dcgm` available?
`defaultgateway`	Is there a default gateway available?
`dellnss`	If running, is the Dell NFS Storage Solution healthy?
`diskspace`	Is there less local disk space available to non-root users than any of the space parameters specified? *The space parameters can be specified as MB, GB, TB, or as percentages with %. The default severity of notices from this check is 10, when one space parameter is used. For more than one space parameter, the severity decreases by 10 for each space parameter, sequentially, down to 10 for the last space parameter. By default a space parameter of 10% is assumed. Another, also optional, non-space parameter, the filesystem mount point parameter, can be specified after the last space parameter to track filesystem space, instead of disk space. A metric-based alternative to tracking filesystem space changes is to use the built-in metric* `freespace` *(page 993) instead.*

...continued

Table G.2.1: List Of Health Checks...continued

Name	**Query** (response is `PASS/FAIL`)
	Examples: • `diskspace 10%` less than 10% space = FAIL, severity 10 • `diskspace 10% 20% 30%` less than 30% space = FAIL, with severity levels as indicated: **space left** / **severity** `10%` / 30 `20%` / 20 `30%` / 10 • `diskspace 10GB 20GB` less than 20GB space = FAIL, severity 10 less than 10GB space = FAIL, severity 20 • `diskspace 10% 20% /var` For the filesystem `/var`: less than 20% space = FAIL, severity 10 less than 10% space = FAIL, severity 20

...continued

Table G.2.1: List Of Health Checks...continued

Name	Query (response is PASS/FAIL)
dmesg	Is dmesg output OK? *Regexes to parse the output can be constructed in the configuration file at /cm/local/apps/cmd/scripts/healthchecks/configfiles/dmesg.py*
docker	Is Docker running OK? Checks for Docker server availability and corruption, dead containers, proper endpoints
dockerregistry	Is the Docker registry running OK? Checks registry endpoint and registry availability
exports	Are all filesystems as defined by the cluster management system exported?
etcd	Are the core etcd processes of Kubernetes running OK? Checks endpoints and interfaces
failedprejob	Are there failed prejob health checks (section 7.8.2)? Here: yes = FAIL. By default, the job ID is saved under /cm/local/apps/<*scheduler*>/var/: • On FAIL, in failedprejobs. • On PASS, in allprejobs The maximum number of IDs stored is 1000 by default. The maximum period for which the IDs are stored is 30 days by default. Both these maxima can be set with the failedprejob health check script.
hpraid	Are the HP Smart Array controllers OK?
ib	Is the InfiniBand Host Channel Adapter working properly? *A configuration file for this health check is at /cm/local/apps/cmd/scripts/healthchecks/configfiles/ib.py*
interfaces	Are the interfaces up and running at full speed?
ipmihealth	Is the BMC (IPMI or iLO) health OK? Uses the script sample_ipmi.
kubernetescertsexpiration	Are Kubernetes certificates valid for at least the next 30 days?

...continued

Table G.2.1: List Of Health Checks...continued

Name	**Query** (response is `PASS/FAIL`)
`kuberneteschildnode`	Are all Kubernetes child nodes up?
`kubernetescomponentsstatus`	Are all expected agents and services up and running for active nodes?
`kubernetesnodesstatus`	Is the status for all Kubernetes nodes OK?
`kubernetespodsstatus`	Is the status for all pods OK?
`ldap`	Can the ID of the user be looked up with LDAP?
`lustre`	Is the Lustre filesystem running OK?
`megaraid`	Are the MegaRAID controllers OK? *Either the proprietary MegaCLI software, or its successor, the proprietary StorCLI software is needed for this health check. The MegaCLI software was originally provided by LSI Logic, but LSI is now part of Broadcom.* *Both the MegaCLI software and the StorCLI software are now available from the Broadcom website (*`http://www.broadcom.com`*).* *For BCM 10 and onwards, the healthcheck first checks for StorCLI, and then for MegaCLI, and uses the first binary that is detected. For BCM versions prior to version 10, the healthcheck first checks for MegaCLI, and then StorCLI, and uses the first binary that is detected.*
`mounts`	Are all mounts defined in the fstab OK?
`mysql`	Is the status and configuration of MySQL correct?
`node-hardware-profile`	Is the specified node's hardware configuration during health check use unchanged? *The options to this script are described using the "-h" help option. Before this script is used for health checks, the specified hardware profile is usually first saved with the* `-s` *option. Eg: "*`node-hardware-profile -n node001 -s hardwarenode001`*"*
`ntp`*	Is NTP synchronization happening?
`oomkiller`	Has the oomkiller process run? Yes=FAIL. The `oomkiller` health check checks if the oomkiller process has run. The configuration file `/cm/local/apps/cmd/scripts/healthchecks/configfiles/oomkiller.conf` for the `oomkiller` health check can be configured to reset the response to PASS after one FAIL is logged, until the next oomkiller process runs. The processes killed by the oomkiller process are logged in `/var/spool/cmd/save-oomkilleraction`. *A consideration of the causes and consequences of the killed processes is strongly recommended. A reset of the node is generally recommended.*

...continued

Table G.2.1: List Of Health Checks...continued

Name	Query (response is PASS/FAIL)
opalinkhealth	Are the quality and the integrity of the Intel OPA HFI link OK?
Overall_Health:<*sda*>	Overall disk health status (SMART response) for specified device, in this case <*sda*>.
SMART_Health:<*sda*>	Overall SMART health as reported by the exit code, for a specified device, in this case <*sda*>.
rogueprocess	Are the processes that are running legitimate (ie, not 'rogue')? Besides the FAIL/PASS/UNKNOWN response to CMDaemon, also returns a list of rogue process IDs to file descriptor 3 (InfoMessages), which the killprocess action (page 1046) can then go ahead and kill. Illegitimate processes are processes that should not be running on the node. An illegitimate process is at least one of the following, by default: • not part of the workload manager service or its jobs • not a root- or system-owned process • in the state Z, T, W, or X. States are described in the ps man pages in the section on "PROCESS STATE CODES"

...continued

Table G.2.1: List Of Health Checks...continued

Name	**Query** (response is `PASS/FAIL`)
	Rogue process criteria can be configured in the file `/cm/local/apps/cmd/scripts/healthchecks/configfiles/rogueprocess.py` within the software image. To implement a changed criteria configuration, the software image used by systems on which the health check is run should be updated (section 5.6). For example, using: `cmsh -c "device; imageupdate -c default -w"` for the `default` category of nodes.
`schedulers`	Are the queue instances of all schedulers on a node healthy ?
`smart`	Is the SMART response healthy? The severities can be configured in the file `/cm/local/apps/cmd/scripts/healthchecks/configfiles/smart.conf`. By default, if a drive does not support the SMART commands and results in a "Smart command failed" info message for that drive, then the healthcheck is configured to give a PASS response. This is because the mere fact that the drive is a non-SMART drive should not be a reason to conclude that the drive is unhealthy. The info messages can be suppressed by setting an allowed list of the disks to be checked within `/cm/local/apps/cmd/scripts/healthchecks/smart`.
`ssh2node`	Is passwordless ssh root login, from head to a node that is up, working? Some details of its behavior are: • The health check fails on the head node if root ssh login to the head node has been disabled. • The health check fails if ssh certificate-based access, to a non-head node that is in the `UP` state, fails, even if login (key-based) access is still available. The `UP` state is determined by whether CMDaemon is running on that node. • If the regular node is in a `DOWN` state—which could be due to CMDaemon being down, or the node having been powered off gracefully, or the node suffering a sudden power failure—then the health check responds with a `PASS`. The idea here is to check key or certificate access, and decouple it from the node state.

...continued

Table G.2.1: List Of Health Checks...continued

Name	Query (response is PASS/FAIL)
swraid	Are the software RAID arrays healthy?
testhealthcheck	*A health check script example for creating scripts, or setting a mix of PASS/FAIL/UNKNOWN responses. The source includes examples of environment variables that can be used, as well as configuration suggestions.*

* built-ins, not standalone scripts.
If sampling from a head node, a standalone script is in directory:
/cm/local/apps/cmd/scripts/healthchecks/
If sampling from a regular node, a standalone script is in directory:
/cm/images/default-image/cm/local/apps/cmd/scripts/healthchecks/

G.2.2 GPU Health Checks

The data producer (section 10.2.10) for the GPU health checks of this section is GPUSampler.

The NVIDIA GPU health checks of this section, as the GPUSampler data producer name suggests, is about gathering the sampled GPU data from the devices themselves.

The device parameter for the GPU health checks in this section, unless otherwise noted, requires as a parameter the device slot number that the GPU uses. The parameter takes the form gpu0, gpu1, and so on. It is appended to the metric with a colon character. For example, the gpu_health_inforom health check, if used with gpu1, is specified as:

Example

```
gpu_health_inforom:gpu1
```

Available GPU health checks for V100 and A100 GPUs are listed in table G.2.2. The available GPU metrics are displayed in table G.1.6.

A DCGM WARN result does not add a fourth state to these health checks — the value reported for a gpu_health_* check remains one of PASS, UNKNOWN, or FAIL. By default, a DCGM WARN is reported as PASS, with the reason recorded in the check's Info field rather than changing its Value:

Example

```
[basecm11->device[dgx2-1]]% latesthealthdata -m gpu_health_overall:gpu0
Measurable          Parameter  Type  Value  Age     State  Info
------------------  ---------  ----  -----  ------  -----  -------------------
gpu_health_overall  gpu0       GPU   PASS   1m 21s         DCGM health warning
```

This default can be overridden per cluster by enabling the Warn as fail extra parameter on the GPUSampler data producer, which makes a WARN result show as FAIL instead of PASS:

Example

```
[basecm11]% monitoring setup
[basecm11->monitoring->setup]% use gpusampler
[basecm11->monitoring->setup[GPUSampler]]% set -e "Warn as fail" true
[basecm11->monitoring->setup*[GPUSampler*]]% commit
```

The Warn as fail extra parameter is false by default. Setting it to true applies to all gpu_health_* checks from the GPUSampler producer, whether specified with a GPU parameter (for example gpu_health_overall:gpu0) or without one.

Table G.2.2: List Of GPU Health Checks

Name	**Query** (script response is `PASS/FAIL`)
`gpu_health_driver`	Is the driver-related subsystem OK?
`gpu_health_hostengine`*	Is the host engine status, for all GPU devices on that node, OK?
`gpu_health_inforom`	Is the Inforom OK?
`gpu_health_mcu`	Is the microcontroller unit OK?
`gpu_health_mem`	Is the memory subsystem OK?
`gpu_health_nvlink`	Is the NVLINK system OK?
`gpu_health_nvswitch_fatal`	Is the NVSwitch showing no fatal errors?
`gpu_health_nvswitch_non_fatal`	Is the NVSwitch showing no non-fatal errors?
`gpu_health_overall`**	Is the overall GPU health OK?
`gpu_health_pcie`	Is the PCIe system OK?
`gpu_health_pmu`	Is the power management unit OK?
`gpu_health_power`	Is the power OK?
`gpu_health_sm`	Is the streaming multiprocessor OK?
`gpu_health_thermal`	Is the temperature OK?

* Specified without a GPU because the check is a check for all GPUs.
** If specified without a GPU, then the check is a check for all GPUs.
For example, for the `gpu_health_overall` health check:
`gpu_health_overall` is for all the GPUs
`gpu_health_overall:gpu0` is just for GPU0

G.2.3 Redfish Health Checks

The available Redfish health checks are displayed in table G.2.3.

Table G.2.3: Redfish health checks

Redfish Health Check	**Description**
`chassis_health`	Health status of chassis
`cpu_health`	Health status of processor
`memory_health`	Health status of memory
`storage_health`	Health status of storage
`device_health`	Health status of storage device
`drive_health`	Health status of storage drive
`volume_health`	Health status of storage volume
`psu_health`	Health status of power supply

...continues

Table G.1.3: Redfish health checks...continued

Redfish Health Check	Description
fan_health	Health status of a fan
sensor_health	Health status of a sensor
pcie_health	Health status of PCIe device
manager_health	Health status of manager (e.g.: HPE iLO)

G.2.4 NetQ Health Checks

NetQ health checks (table G.2.4) are healthcheck measurables sourced from NetQ. Configuring NetQ with BCM is described in section 3.11.

The data producer (section 10.2.10) for NetQ measurables is the netq data producer.

Table G.2.4: NetQ Health Checks

NetQ Health Check	**Query** (Response is PASS/FAIL)
NetQ_node_fan_status	Is the fan working?
NetQ_node_NVLink_status	Is NVLink up?
NetQ_node_PSU_status	Is the power supply unit working?
NetQ_node_Temp_status	Is the temperature sensor working?

G.2.5 Parameters For Health Checks

Health checks have the parameters indicated by the left column in the example below:

Example

```
[myheadnode->monitoring->measurable]% show cmsh
Parameter                      Value
------------------------------ ------------------------------------------------
Class                          Internal
Consolidator                   - (cmsh)
Description                    Checks whether cmsh is available, i.e. can we
                               use cmsh for the default cluster?
Disabled                       no (cmsh)
Gap                            0 (cmsh)
Maximal age                    0s (cmsh)
Maximal samples                4,096 (cmsh)
Name                           cmsh
Parameter
Producer                       cmsh
Revision
Type                           HealthCheck
```

If the value is inherited from the producer, then it is shown in parentheses next to the value. An inherited value can be overwritten by setting it directly for the parameter of a measurable.
The parameters are a subset of the parameters for metrics described in section G.1.15.

G.3 Enummetrics

Table G.3: List Of Enummetrics

Name	Query
`DeviceStatus`	What is the status of the device? Possible values are: • `up` • `down` • `pending` • `installing` • `installer_failed` • `installer_rebooting,` • `installer_calling_init` • `installer_unreachable` • `going_down` • `booting` • `undefined` • `closed` • `muted` • `terminated` • `burning` • `unknown` • `opening` • `no data`

...continued

Table G.3: List Of Enummetrics...continued

Name	Query
`wlm_slurm_state`	What is the status of the device allocated to Slurm? Possible values are: • `allocated` • `completing` • `down` • `drain` • `draining` • `fail` • `failing` • `maint` • `idle` • `mixed`

G.4 Actions And Their Parameters

G.4.1 Actions

Table G.4.1: List Of Actions

Name	Description
Drain	Allows no new processes on a compute node from the workload manager. This means that already running jobs are permitted to complete. Usage Tip: Plan for undrain from another node becoming active
Send e-mail to administrators	Sends mail using the mailserver that was set up during server configuration. Default destination is root@localhost. The e-mail address that it is otherwise sent to is specified by the recipient parameter for this action.
Event	Send an event to users with a connected client
ImageUpdate	Update the image on the node
PowerOff	Powers off, hard
PowerOn	Powers on, hard
PowerReset	Power reset, hard
Reboot	Reboot via the system, trying to shut everything down cleanly, and then start up again
killprocess*	Kills processes seen by CMDaemon with the KILL (-9) signal. The PIDs are passed via the CMD_INFO_MESSAGE environmental variable. Syntax: killprocess <*PID1*[,<*PID2*>,...]> This action is designed to work with rogueprocess (page 1039)
remount*	remounts all defined mounts
testaction*	An action script example for users who would like to create their own scripts. The source has helpful remarks about the environment variables that can be used as well as tips on configuring it generally
Shutdown	Power off via system, trying to shut everything down cleanly
Undrain node	Allow processes to run on the node from the workload manager

* standalone scripts, not built-ins.
If running from a head node, the script is in directory: /cm/local/apps/cmd/scripts/actions/
If running from a regular node, the script is in directory: /cm/images/default-image/cm/local/apps/cmd/scripts/actions/

G.4.2 Parameters For A Monitoring Action

The default monitoring actions are listed in section 10.4.4.

All actions have in common the parameters shown by the left column, illustrated by the example below for the drain action:

Example

```
[myheadnode->monitoring->action]% show drain
Parameter          Value
------------------ ----------------------
Action             Drain node from all WLM
Allowed time
Disable            no
Name               Drain
Revision
Run on             Active
Type               DrainAction
```

Out of the full list of default actions, the actions with only the common parameter settings are:

- `Poweron`: Powers off the node
- `PowerOff`: Powers off the node
- `PowerReset`: Hard resets the node
- `Drain`: Drains the node (does not allow new jobs on that node)
- `Undrain`: Undrains the node (allows new jobs on that node)
- `Reboot`: Reboots node via the operating system.
- `Shutdown`: Shuts the node down via the operating system.
- `ImageUpdate`: Updates the node from the software image
- `Event`: Sends an event to users connected with `cmsh` or Base View

Extra Parameters For Some Actions

The following actions have extra parameters:

Action of the type `ScriptAction`:

- `killprocess`: A script that kills a specified process
- `testaction`: A test script
- `remount`: A script to remount all devices

The extra parameters for an action of type `ScriptAction` are:

  - `Arguments`: List of arguments that are taken by the script
  - `Node environment`: Does the script run in the node environment?
  - `Script`: The script path
  - `timeout`: Time within which the script must run before giving up

Action of the type `EmailAction`:

- `Send e-mail to administrators`: Sends an e-mail out, by default to the administrators

The extra parameters for an action of type `EmailAction` are:

  - `All administrators`: sends the e-mail to the list of users in the `Administrator e-mail` setting in `partition[base]` mode
  - `Info`: the body of the e-mail message
  - `Recipients`: a list of recipients

H

Workload Manager Configuration Files Updated By CMDaemon

This appendix lists workload manager configuration files changed by CMDaemon, events causing such change, and the file or property changed.

H.1 Slurm

File/Property	Updates What?	Updated During
/cm/shared/apps/slurm/var/etc/<*Slurm instance name*>/slurm.conf	head node	Add/Remove/Update nodes, hostname change
/cm/shared/apps/slurm/var/etc/<*Slurm instance name*>/slurmdbd.conf	head node	Add/Remove/Update nodes, hostname change
/cm/shared/apps/slurm/var/etc/<*Slurm instance name*>/gres.conf	all nodes	Add/Remove/Update nodes
/cm/shared/apps/slurm/var/etc/<*Slurm instance name*>/topology.conf	head node	Add/Remove/Update nodes

H.2 LSF

File/Property	Updates What?	Updated During
$LSF_ENVDIR/lsf.conf	head node	hostname/domain change, failover
$LSF_ENVDIR/lsf.cluster.<*clustername*>	head node	add/remove/update nodes
$LSF_ENVDIR/lsf.sudoers	head node	hostname/domain change, failover
$LSF_ENVDIR/hosts	cloud-director	add/remove/update cloud nodes
$LSF_ENVDIR/lsbatch/<*clustername*>/configdir/lsb.queues	head node	add/remove/update queues

...continues

...continued

File/Property	Updates What?	Updated During
`$LSF_ENVDIR/lsbatch/<clustername>/configdir/lsb.hosts`	head node	add/remove/update nodes

The default value of $LSF_ENVDIR in BCM is `/cm/shared/apps/lsf/var/conf/<clustername>`

On each node where the `lsfd` service runs, CMDaemon creates a symlink `/etc/lsf.conf` that points to `$LSF_ENVDIR/lsf.conf`. This is required by LSF daemons.

I

Changing The LDAP Password

The administrator may wish to change the LDAP root password. This procedure has two steps:

- setting a new password for the LDAP server (section I.1), and
- setting the new password in `cmd.conf` (section I.2).

It is also a good idea to do some checking afterwards (section I.3).

I.1 Setting A New Password For The LDAP Server

An encrypted password string can be generated as follows:

```
[root@basecm11 ~]# module load openldap
[root@basecm11 ~]# slappasswd
New password:
Re-enter new password:
SSHAJ/3wyO+IqyAwhh8Q4obL8489CWJlHpLg
```

The input is the plain text password, and the output is the encrypted password. The encrypted password is set as a value for the `rootpw` tag in the `slapd.conf` file on the head node:

```
[root@basecm11 ~]# grep ^rootpw /cm/local/apps/openldap/etc/slapd.conf
rootpw SSHAJ/3wyO+IqyAwhh8Q4obL8489CWJlHpLg
```

The password can also be saved in plain text instead of as an SSHA hash generated with `slappasswd`, but this is considered insecure.

After setting the value for `rootpw`, the LDAP server is restarted:

```
[root@basecm11 ~]# systemctl restart slapd
```

I.2 Setting The New Password In `cmd.conf`

The new LDAP password (the plain text password that generated the encrypted password after entering the `slappasswd` command in section I.1) is set in `cmd.conf`. It is kept as clear text for the entry for the LDAPPass directive (Appendix C):

```
[root@basecm11 ~]# grep LDAPPass /cm/local/apps/cmd/etc/cmd.conf
LDAPPass = "Mysecret1dappassw0rd"
```

CMDaemon is then restarted:

```
[root@basecm11 ~]# systemctl restart cmd
```

I.3 Checking LDAP Access

For a default configuration with user `cmsupport` and domain `cm.cluster`, the following checks can be run from the head node (some output truncated):

- anonymous access:

```
[root@basecm11 ~]# ldapsearch -x
# extended LDIF
#
# LDAPv3
# base <dc=cm,dc=cluster> (default) with scope subtree
...
```

- root cn without a password (this should fail):

```
[root@basecm11 ~]# ldapsearch -x -D 'cn=root,dc=cm,dc=cluster'
ldap_bind: Server is unwilling to perform (53)
 additional info: unauthenticated bind (DN with no password) disallowed
[root@basecm11 ~]#
```

- root cn with a password (this should work):

```
[root@basecm11 ~]# ldapsearch -x -D 'cn=root,dc=cm,dc=cluster' -W
Enter LDAP Password: (the password entered earlier, Mysecret1dappassw0rd, is used)
# extended LDIF
#
# LDAPv3
# base <dc=cm,dc=cluster> (default) with scope subtree
...
```

J

Tokens

This appendix describes authorization tokens available for profiles. Profiles are introduced in Section 6.5:

Useful for listing services and tokens are the following `cmsh` commands, available within the `profile` mode:

- `allservices`
- `alltokens`
- `showservices`

Example

```
[root@basecm11 ~]# cmsh
[basecm11]% profile
[basecm11->profile]% allservices
auth
cert
cloud
device
etcd
gui
job
kube
main
mon
net
part
proc
prov
serv
session
status
test
user
```

Similarly, the output from `showservices` displays the tokens for the services as indicated in the following table:

Table J: List Of Tokens

Service and token name	User can...
Service: CMAuth	
ADD_PROFILE_TOKEN	Add a new profile
GET_CMSERVICES_TOKEN	Get a list of available CMDaemon services
GET_PROFILE_TOKEN	Retrieve list of profiles and profile properties
UPDATE_PROFILE_TOKEN	Update profile
Service: CMCert	
GET_CERTIFICATE_INFORMATION_TOKEN	Get certificate information
GET_CERTIFICATE_INFO_TOKEN	Get certificate information
GET_CERTIFICATE_REQUEST_TOKEN	List pending certificate requests
GET_CERTIFICATE_TOKEN	Get certificate
INVALIDATE_COMPONENT_CA_TOKEN	Invalidate component CA
TOKEN	
ISSUE_CERTIFICATE_TOKEN	Accept certificate request and issue signed certificate
RECREATE_COMPONENT_CERTIFICATE_ TOKEN	Recreate component certificate
REMOVE_CERTIFICATE_REQUEST_TOKEN	Cancel certificate request
REMOVE_CERTIFICATE_TOKEN	Remove a certificate
REVOKE_CERTIFICATE_TOKEN	Revoke a certificate
UNREVOKE_CERTIFICATE_TOKEN	Unrevoke a revoked certificate
Service: CMCloud	
ADD_CLOUD_JOB_DESCRIPTION_TOKEN	Add cloud job description
ADD_CLOUD_PROVIDER_TOKEN	Add a new cloud provider
ADD_OCI_INSTANCE_POOL_TOKEN	Add OCI instance pool
AZURE_ACCESS_STRING_TOKEN	Get/set Azure access string
CANCEL_ANY_CLOUD_JOB_TOKEN	Cancel any cloud job
CLOUD_DIRECTOR_NEW_IP_TOKEN	Set the new External IP of the cloud director
DELETE_ANY_ANF_VOLUME_TOKEN	Delete any ANF volume
DELETE_ANY_FSX_INSTANCE_TOKEN	Delete any FSX instance
EC2_ACCESS_STRING_TOKEN	Get/set Amazon EC2 access string
FORGE_ACCESS_STRING_TOKEN	Forge an access string
GET_ALL_CLOUD_JOB_DESCRIPTION_TOKEN	Get all cloud job description
GET_CLOUD_AMI_TOKEN	Access Amazon EC2 AMI

...continues

Table J: List Of Tokens...continued

Service and token name	User can...
GET_CLOUD_JOB_DESCRIPTION_TOKEN	Get cloud job description
GET_CLOUD_PROVIDER_TOKEN	Get cloud provider information
GET_CLOUD_REGION_TOKEN	Access Amazon EC2 region
GET_CLOUD_TYPE_TOKEN	Access Amazon instance type
GET_CONSOLE_OUTPUT_TOKEN	Retrieve the console output of the cloud director for debugging purposes
GET_KERNEL_INITRD_MD5SUM_TOKEN	Retrieve MD5 sum of initial ramdisk
GET_OCI_INSTANCE_POOL_TOKEN	Get OCI instance pool
LIST_ALL_ANF_VOLUMES_TOKEN	List all ANF volumes
LIST_ALL_FSX_INSTANCES_TOKEN	List all FSX instances
OCI_ACCESS_STRING_TOKEN	OCI access string
ON_DEMAND_ANF_TOKEN	On demand ANF
ON_DEMAND_FSX_TOKEN	On demand FSX
OSCLOUD_ACCESS_STRING_TOKEN	Get/set OpenStack access string
PUT_USERDATA_TOKEN	Set AWS user data in AWS
SEND_CLOUD_STORAGE_ACTION_TOKEN	Send cloud storage action
SET_CLOUDERRORS_TOKEN	Set cloud errors
SHARE_ANF_VOLUME_TOKEN	Share own ANF volumes
SHARE_FSX_INSTANCE_TOKEN	Share own FSX instances
SUBMIT_CLOUD_JOB_DESCRIPTION_TOKEN	Submit cloud job description
TERMINATE_NODE_TOKEN	Terminate cloud nodes
UPDATE_CLOUD_JOB_DESCRIPTION_TOKEN	Update cloud job description
UPDATE_CLOUD_PROVIDER_TOKEN	Update cloud provider settings
UPDATE_OCI_INSTANCE_POOL_TOKEN	Update OCI instance pool
USER_MANAGED_ANF_TOKEN	Manage ANF volume
USER_MANAGED_FSX_TOKEN	Manage FSX volume
Service: CMDevice	
ACCESS_SETTINGS_TOKEN	Access settings
ADD_CATEGORY_TOKEN	Create new category
ADD_CONFIGURATIONOVERLAY_TOKEN	Create new configuration overlay
ADD_DEVICE_TOKEN	Add a new device
ADD_FILE_WRITE_INFO_TOKEN	Add filewriteinfo
ADD_LITENODE_TOKEN	Add lite node
ADD_NODEGROUP_TOKEN	Add a new nodegroup
ADD_NODE_HIERARCHY_RULE_TOKEN	Add node hierarchy rule
ADD_REMOTE_NODE_INSTALLER_INTERACTION_TOKEN	Add a node-installer interaction (Used by CMDaemon)
ADD_REPORT_QUERY_TOKEN	Add report query
ADD_GPU_WORKLOAD_QUERY_	Add GPU workload query performance

...continues

Table J: List Of Tokens...continued

Service and token name	User can...
PERFORMANCE_PROFILE_TOKEN	profile
APPLY_DEVICE_COMMANDS_TOKEN	Apply device commands token
APPLY_PTM_TOPOLOGY_TOKEN	Apply PTM Topology
BIOS_APPLY_TOKEN	BIOS apply token
BIOS_FETCH_TOKEN	BIOS fetch token
BMC_USERNAME_PASSWORD_TOKEN	View/set BMC (e.g. HP ilo4, IPMI) username and password
BURN_STATUS_TOKEN	Get burn status
CANCEL_BURN_TOKEN	Cancel burn token
CHANGED_NVDOMAIN_INFO_TOKEN	Changed NVIDIA domain info
CHASSIS_USER_PASSWORD_TOKEN	Get/set chassis username and password
CHECK_REMOTE_MOUNT_TOKEN	Check remount mount
CLEAR_DISK_ENCRYPTION_PASSPHRASE_TOKEN	Clear disk encryption passphrase
COMPLETE_BURN_TOKEN	Complete burn
CREATE_PORT_FORWARD_BURN_TOKEN	Complete port forwarding
DIFF_DEVICE_COMMANDS_TOKEN	Diff device commands
DIRTY_GPU_WORKLOAD_PERFORMANCE_PROFILE_CACHE_TOKEN	Dirty GPU workload performance profile cache
FETCH_NVDOMAIN_INFO_TOKEN	Fetch NVIDIA domain info
FIRMWARE_FLASH_TOKEN	Flash firmware
FIRMWARE_INFO_TOKEN	Get firmware information
FIRMWARE_UPLOAD_TOKEN	Upload firmware
FORCE_RECONNECT_TOKEN	Force reconnect
GET_BACKUP_DEVICE_COMMANDS_TOKEN	Get backup device commands
GET_BACKUP_INFO_TOKEN	Get backup information
GET_BURN_LOG_TOKEN	Retrieve burn log
GET_BURN_TOKEN	Get burn token
GET_CATEGORY_TOKEN	Get list of categories
GET_CONFIGURATIONOVERLAY_TOKEN	Get list of configuration overlays
GET_DEVICE_BY_PORT_TOKEN	View list of devices according to the ethernet switch port that they are connected to
GET_DEVICE_COMMANDS_TOKEN	Get device commands
GET_DEVICE_LEAK_INFO_TOKEN	Get device leak info
GET_DEVICE_TOKEN	View all device properties
GET_DEVICE_UUID_TOKEN	View device UUID
GET_DHCPD_LEASES_TOKEN	View device DHCPD leases
GET_DISKSETUP_TOKEN	Get disksetup
GET_DPU_TOKEN	Get DPU
GET_EXCLUDE_LIST_TOKEN	Retrieve the various exclude lists
GET_FILE_WRITE_INFO_TOKEN	Get filewriteinfo

...continues

Table J: List Of Tokens...continued

Service and token name	User can...
GET_FINALIZE_SCRIPTS_TOKEN	Get finalize scripts
GET_FREE_PORTS_TOKEN	Get free ports info
GET_GPU_MIG_TOKEN	Get GPU MIG
GET_GPU_PROFILING_METRIC_INFO_TOKEN	Get GPU profiling metric info
GET_GPU_PROFILING_STATE_TOKEN	Get GPU profiling state
GET_GPU_WORKLOAD_PERFORMANCE_PROFILE_TOKEN	View GPU workload performance profile
GET_IBSWITCH_TOKEN	View IB switch properties
GET_IMEX_CTL_TOKEN	View IMEX_CTL properties
GET_INITIALIZE_SCRIPTS_TOKEN	Get initialize scripts
GET_MINIMAL_CONFIG_TOKEN	View minimum configuration
GET_NETWORK_TOPOLOGY_TOKEN	Get network topology
GET_NODEGROUP_TOKEN	Get list of nodegroups
GET_NODE_ACCELERATOR_TOKEN	Get node accelerator count
GET_NODE_ARCH_OS_TOKEN	Get architecture and OS
GET_NODE_HIERARCHY_RULE_TOKEN	Get node hierarchy rule
GET_NVDOMAIN_INFO_TOKEN	Get NVIDIA domain info
GET_NVLINK_INFO_TOKEN	Get NVIDIA link info
GET_PORT_BY_MAC_TOKEN	Determine to which switch port a given MAC is connected to.
GET_PTM_TOPOLOGY_TOKEN	Get PTM Topology
GET_REMOTE_NODE_INSTALLER_INTERACTIONS_TOKEN	Get list of pending installer interactions
GET_REPORT_QUERY_TOKEN	Get report query
GET_SCRIPT_ENVIRONMENT_TOKEN	Get script environment
GET_SWITCH_COMMAND_TEMPLATES_TOKEN	Get switch command templates
GET_SWITCH_FIRMWARES_TOKEN	Get switch firmwares
GET_SWITCH_IMAGES_TOKEN	Get switch images
GET_SWITCH_ZTP_TEMPLATES_TOKEN	Get switch ZTP templates
GET_SYNC_INFO_TOKEN	Get rsync information
GET_SYNC_LOG_TOKEN	Get rsync provisioning log
GET_SYSINFO_COLLECTOR_TOKEN	Get information about a node (executes dmidecode)
GET_TFTPBOOT_FILE_INFORMATION_TOKEN	Get tftpboot firmware information
GET_USED_PORTS_TOKEN	Get used ports info
GET_WIREGUARD_INFO_TOKEN	Get wireguard info
LIST_DEVICE_COMMANDS_TOKEN	List device commands
LIST_DPU_BFB_TOKEN	List DPU BFB
LIST_IBSWITCH_TOKEN	List IB switches
LIST_PORT_FORWARD_TOKEN	List port forwarding

...continues

Table J: List Of Tokens...continued

Service and token name	User can...
MALLOC_TRIM_TOKEN	malloc trim
NEW_NODE_TOKEN	New node
NODE_GET_MOUNTPOINTS_TOKEN	Get list of mountpoints defined for a node
NODE_IDENTIFY_TOKEN	Identify a node (RPC API, used by node installer)
NVSM_ALERTS_TOKEN	View NVSM alerts
NVSM_HEALTH_TOKEN	View NVSM health
NVSM_INFO_TOKEN	View NVSM info
NVSM_VERSIONS_TOKEN	View NVSM versions
NV_FABRIC_INFO_TOKEN	View NVIDIA fabric info
NV_FABRIC_START_STOP_TOKEN	Carry out NVIDIA fabric start and stop
NV_FABRIC_START_TOKEN	Carry out NVIDIA fabric start
NV_FABRIC_STATUS_TOKEN	View NVIDIA fabric status
NV_FABRIC_STOP_TOKEN	Carry out NVIDIA fabric stop
PMC_USERNAME_PASSWORD_TOKEN	View/set Power management controller username and password
POWER_CANCEL_TOKEN	Power cancel operation
POWER_CYCLE_TOKEN	Power reset a device
POWER_OFF_TOKEN	Power off a device
POWER_ON_TOKEN	Power on a device using BMC or PDU power control
POWER_STATUS_TOKEN	Get power status e.g on or off
PPING_TOKEN	Run parallel ping
PREPARE_POWER_OFF_TOKEN	Prepare to power off a device
PROXY_SETTINGS_TOKEN	Proxy settings
PUSH_DPU_BFB_TOKEN	Push DPU BPB
PUT_SYSINFO_COLLECTOR_TOKEN	Put information about a node
REBOOT_NODE_TOKEN	Reboot a remote a node
REDFISH_EVENT_TOKEN	Redfish event
REFRESH_NVDOMAIN_INFO_TOKEN	Refresh NVIDIA domain info
REMOVE_BACKUP_TOKEN	Remove backup information
REMOVE_REMOTE_NODE_INSTALLER_INTERACTION_TOKEN	Remove a node installer interaction
REMOVE_SYSINFO_COLLECTOR_TOKEN	Remove information about a node
REPORT_POWER_STATUS_TOKEN	Report power operation history
REPORT_QUERY_TOKEN	Show report query
REQUEST_BURN_TOKEN	Request burn
RESET_GPU_TOKEN	Reset GPU
RUN_POST_CHANGE_ACTIONS_TOKEN	Run POST change actions
SET_BACKUP_INFO_TOKEN	Set backup information
SET_DEVICE_LEAK_INFO_TOKEN	Set device leak info
SET_POWER_CONFIG_TOKEN	Set device power configuration

...continues

Table J: List Of Tokens...continued

Service and token name	User can...
SET_DEVICE_STATUS_TOKEN	Set device status (only via RPC API calls)
SET_DPU_TOKEN	Set DPU
SET_GPU_MIG_TOKEN	Set GPU MIG
SET_GPU_WORKLOAD_PERFORMANCE_PROFILE_TOKEN	Set GPU workload performance profile
SET_NODE_ACCELERATOR_TOKEN	Set node accelerator
SET_NODE_ARCH_OS_TOKEN	Set architecture and OS
SET_NVDOMAIN_INFO_TOKEN	Set NVIDIA domain info
SHOW_DEVICE_COMMANDS_TOKEN	Show device commands
SHUTDOWN_NODE_TOKEN	Shutdown a remote node managed by CMDaemon
SNMP_SETTINGS_TOKEN	Manage SNMP settings
START_BURN_TOKEN	Start burn
STOP_BURN_TOKEN	Stop burn
SYSINFO_COLLECTOR_TOKEN	Manage information about a node
TAKE_BACKUP_DEVICE_COMMANDS_TOKEN	Take backup device commands
UPDATE_CATEGORY_TOKEN	Update a category property
UPDATE_CONFIGURATIONOVERLAY_TOKEN	Update a configuration overlay property
UPDATE_DEVICE_TOKEN	Update device properties
UPDATE_GPU_PROFILING_METRIC_INFO_TOKEN	Update GPU profiling metric info
UPDATE_GPU_PROFILING_STATE_TOKEN	Update GPU profiling state
UPDATE_LITENODE_TOKEN	Update lite node
UPDATE_NODEGROUP_TOKEN	Update nodegroup properties (e.g. add a new member node)
UPDATE_NODE_HIERARCHY_RULE_TOKEN	Update node hierarchy rule
UPDATE_REMOTE_NODE_INSTALLER_INTERACTIONS_TOKEN	Update installer interactions (e.g. confirm full provisioning)
UPDATE_REPORT_QUERY_TOKEN	Update report query
UPDATE_STATUS_TOKEN	Update status
UPDATE_SWITCH_TOKEN	Update switch
UPDATE_SYSINFO_COLLECTOR_TOKEN	Update information about a node
UPGRADE_IBSWITCH_TOKEN	Carry out IB switch upgrade
Service: CMEtcd	
ADD_ETCD_TOKEN	Add etcd
GET_ETCD_TOKEN	Get etcd
UPDATE_ETCD_TOKEN	Update etcd
Service: CMGui	
EXPAND_COLLAPSE_TOKEN	Get cluster overview

...continues

Table J: List Of Tokens...continued

Service and token name	User can...
GET_CDU_OVERVIEW_TOKEN	Get CDU overview
GET_CLUSTER_OVERVIEW_TOKEN	Get cluster overview
GET_KUBE_OVERVIEW_TOKEN	Get kube cluster overview
GET_NODE_OVERVIEW_TOKEN	Get node overview
GET_NODE_STATUS_TOKEN	Get node status
GET_PDU_OVERVIEW_TOKEN	Get PDU overview
GET_POWER_CIRCUIT_OVERVIEW_TOKEN	Get power circuit overview
GET_POWER_SHELF_OVERVIEW_TOKEN	Get power shelf overview
GET_RACK_OVERVIEW_TOKEN	Get rack overview
GET_SWITCH_OVERVIEW_TOKEN	Get switch overview
Service: CMJob	
ADD_CHARGE_BACK_REQUEST_TOKEN	Add chargeback request
ADD_JOBQUEUE_TOKEN	Add a new job queue
ADD_WLM_CLUSTER_TOKEN	Add WLM cluster
CHARGE_BACK_BY_KEY_TOKEN	Show chargeback by key
CHARGE_BACK_TOKEN	Show chargeback
CHECK_NODE_ALLOCATION_TOKEN	Check node allocation
DRAIN_OVERVIEW_TOKEN	Obtain list of drained nodes
DRAIN_TOKEN	Drain a node
FLUSH_JOB_INFO_TOKEN	Flush job info
GET_CHARGE_BACK_REQUEST_TOKEN	Get chargeback request
GET_JOBINFO_TOKEN	Get job information
GET_JOBQUEUE_TOKEN	Retrieve list of job queues and properties
GET_JOB_PID_GPUS_INFO_TOKEN	Get job PID GPUs info
GET_JOB_TOKEN	Get list of jobs that are currently running
GET_OWN_JOBINFO_TOKEN	Get own job information
GET_OWN_JOB_TOKEN	Get list of own jobs that are currently running
GET_PE_TOKEN	Get list of SGE parallel environments
GET_TOPOLOGY_TOKEN	Get topology
GET_TRACKED_JOBS_TOKEN	Get tracked jobs
GET_WLM_CLUSTER_TOKEN	Get WLM cluster
GET_WLM_POWER_SAVING_TOKEN	Get WLM power saving status
HOLD_JOB_TOKEN	Place a job on hold
HOLD_OWN_JOB_TOKEN	Place own job on hold
JOB_NODE_GRID_TOKEN	Show job node grid
JOB_STARTED_ENDED_TOKEN	Show job started/ended
NOTIFY_JOB_END_TOKEN	Notify job end
RELEASE_JOB_TOKEN	Release a held job

...continues

Table J: List Of Tokens...continued

Service and token name	User can...
RELEASE_OWN_JOB_TOKEN	Release own job
REMOVE_JOBINFO_TOKEN	Remove job information
REQUEUE_JOB_TOKEN	Requeue a job
REQUEUE_OWN_JOB_TOKEN	Requeue own job
RESUME_JOB_TOKEN	Resume suspended job
RESUME_OWN_JOB_TOKEN	Resume own job
SET_PERSISTENT_JOBINFO_TOKEN	Set persistent job information
SUBMIT_JOB_TOKEN	Submit a job using JSON
SUSPEND_JOB_TOKEN	Suspend a job
SUSPEND_OWN_JOB_TOKEN	Suspend own job
UPDATE_CHARGE_BACK_REQUEST_TOKEN	Update chargeback request
UPDATE_JOBQUEUE_TOKEN	Modify job queues
UPDATE_JOB_TOKEN	Update job run-timer parameters
UPDATE_OWN_JOB_TOKEN	Update own job
UPDATE_TOPOLOGY_TOKEN	Update topology
UPDATE_WLM_CLUSTER_SERVER_TOKEN	Update WLM cluster server
UPDATE_WLM_CLUSTER_TOKEN	Update WLM cluster
Service: CMKube	
ADD_KUBE_TOKEN	Add Kube
DRAIN_KUBE_OVERVIEW_TOKEN	Drain Kube overview
DRAIN_KUBE_TOKEN	Drain Kube
GET_CAPI_IMAGE_VERSIONS_TOKEN	Get CAPI image versions
GET_CAPI_TOKEN	Get CAPI
GET_KUBE_JOIN_TOKEN	Get Kube join
GET_KUBE_TOKEN	Get Kube
KUBE_MANAGED_LABELS_TOKEN	Get Kube managed labels response
UPDATE_CAPI_IMAGE_VERSIONS_TOKEN	Update CAPI image versions
UPDATE_CAPI_TOKEN	Update CAPI
UPDATE_KUBE_TOKEN	Update Kube
Service: CMMain	
CANCEL_BACKGROUND_TASKS_TOKEN	Cancel background tasks
CMDAEMON_FAILOVER_TOKEN	Set CMDaemon failover condition achieved
CM_SETUP_EXECUTE_TOKEN	Execute
CM_SETUP_GET_EXECUTION_TOKEN	Get execution
CM_SETUP_REMOVE_EXECUTION_TOKEN	Remove execution
GENERIC_CALL_TOKEN	Make a generic call
GET_ALL_ACTIVE_PASSIVE_UP_KEYS_TOKEN	Get keys for all active and passive nodes that are up

...continues

Table J: List Of Tokens...continued

Service and token name	User can...
GET_BACKGROUND_TASKS_TOKEN	Get background tasks
GET_CLUSTER_SETUP_TOKEN	Get cluster configuration
GET_CONFIG_TOKEN	Get configuration
GET_FROZEN_FILES_TOKEN	Get frozen files
GET_HARDWARE_OVERVIEW_TOKEN	Get frozen files
GET_LICENSE_INFO_TOKEN	Retrieve information about BCM license
GET_SERVER_STATUS_TOKEN	Head node status (e.g. ACTIVE, BECOMEACTIVE etc.)
GET_SERVICESTATE_TOKEN	Get the state of a service
GET_VERSION_TOKEN	Get CMDaemon version and revision
GET_XSD_SCHEMA_TOKEN	Get XSD schema
IMPORT_ENTITY_TOKEN	Import an entity
PING_TOKEN	TCP SYN ping managed devices
REPORT_CRITICAL_ERROR_TOKEN	View critical error report
SAVE_FILE_TOKEN	Save a file on a remote node
SET_SERVICESTATE_TOKEN	Set the state of a service
STORE_CONFIG_FILE_VERSION_TOKEN	Store config file version
STORE_LDAP_CERTIFICATES_TOKEN	Store LDAP certificates
STORE_PRS_CERTIFICATES_TOKEN	Store PRS certificates
Service: CMMon	
ADD_ENTITY_MEASURABLE_TOKEN	Add entity measurable
ADD_LABELED_ENTITY_TOKEN	Add labeled entity
ADD_MONITORING_ACTION_TOKEN	Add monitoring action
ADD_MONITORING_DATA_PRODUCER_TOKEN	Add monitoring data producer
ADD_MONITORING_MEASURABLE_TOKEN	Add monitoring measurable
ADD_MONITORING_STANDALONE_TOKEN	Add monitoring standalone
ADD_MONITORING_TRIGGER_TOKEN	Add monitoring trigger
ADD_PROMETHEUS_QUERY_TOKEN	Add Prometheus query
BACKUP_INFORMATION_TOKEN	Show backup information
CREATE_MONITORING_MEASURABLE_TOKEN	Create monitoring measurable
DROP_MONITORING_DATA_TOKEN	Drop monitoring data
EXECUTE_QUERY_BY_KEY_TOKEN	Execute Prometheus query by key
EXECUTE_QUERY_TOKEN	Execute Prometheus query
FETCH_CACHE_TOKEN	Fetch the cache
GET_DYNAMIC_RESOURCES_TOKEN	Get a dynamic resource
GET_ENTITY_MEASURABLE_TOKEN	Get entity measurable
GET_INFO_MESSAGE_TOKEN	Get info message
GET_LABELED_ENTITY_TOKEN	Get labeled entity
GET_MONITORING_ACTION_TOKEN	Get monitoring action

...continues

Table J: List Of Tokens...continued

Service and token name	User can...
GET_MONITORING_CONSOLIDATOR_TOKEN	Get monitoring consolidator
GET_MONITORING_DATA_PRODUCER_TOKEN	Get monitoring data producer
GET_MONITORING_MEASURABLE_TOKEN	Get monitoring measurable
GET_MONITORING_STANDALONE_TOKEN	Get monitoring standalone
GET_MONITORING_TRIGGER_TOKEN	Get monitoring trigger
GET_PICKUP_INTERVAL_TOKEN	Get pickup interval
GET_PROMETHEUS_JOB_EXTRA_LABEL_CACHE_TOKEN	Get Prometheus extra label cache
GET_PROMETHEUS_QUERY_TOKEN	Get Prometheus query
GET_TRIGGER_DATA_TOKEN	Get monitoring trigger evaluation data
HEALTH_CHECK_WLM_JOB_TOKEN	Manage health checks for WLM job
INTERNAL_DROP_MONITORING_DATA_TOKEN	Internal dropmonitoringdata
INTERNAL_RUN_ACTION_TOKEN	Internal run action
INTERNAL_SAMPLE_NOW_TOKEN	Internal samplenow
MONITORING_CLEANUP_TOKEN	Monitoring cleanup
MONITORING_INFO_TOKEN	Monitoring info
MONITORING_LITE_TOKEN	Monitoring lite node
MONITORING_MANAGE_TOKEN	Manage monitoring configuration settings
MONITORING_PREPARE_CONTINUE_BACKUP_TOKEN	Monitor prepare continue backup
MONITORING_PUSH_TOKEN	Monitoring push
MONITORING_TREE_DEFAULT_SHOW_TOKEN	Monitoring tree default show
NEW_LABELED_ENTITY_TOKEN	Use new labeled entity
NEW_MEASURABLE_TOKEN	New measurable token
OFFLOAD_INFORMATION_TOKEN	Get offload information
PLOT_TOKEN	Request plot
PRIVATE_MONITORING_TOKEN	Private monitoring
PROMETHEUS_EXPORTER_TOKEN	Show Prometheus exporter
PROMETHEUS_METRIC_TOKEN	Show Prometheus metric
PUT_OFFLOAD_INFORMATION_TOKEN	Put offload information
REINITIALIZE_TOKEN	Reinitialize data producers
REQUEST_PICKUP_INTERVAL_TOKEN	Request monitoring pickup interval
SAMPLE_NOW_TOKEN	Sample now
STATE_TRANSITION_TOKEN	State transition
UPDATE_DYNAMIC_RESOURCES_TOKEN	Update a dynamic resource
UPDATE_LABELED_ENTITY_TOKEN	Update labeled entity
UPDATE_MONITORING_ACTION_TOKEN	Update monitoring action
UPDATE_MONITORING_CONSOLIDATOR_TOKEN	Update monitoring consolidator
UPDATE_MONITORING_DATA_PRODUCER_TOKEN	Update monitoring data producer
UPDATE_MONITORING_MEASURABLE_TOKEN	Update monitoring measurable

...continues

Table J: List Of Tokens...continued

Service and token name	User can...
UPDATE_MONITORING_STANDALONE_TOKEN	Update monitoring standalone
UPDATE_MONITORING_TRIGGER_TOKEN	Update monitoring trigger
UPDATE_PROMETHEUS_JOB_EXTRA_LABEL_CACHE_TOKEN	Update Prometheus job extra label cache
UPDATE_PROMETHEUS_QUERY_TOKEN	Update Prometheus query
Service: CMNet	
ADD_NETWORK_TOKEN	Add network settings
GET_NETWORK_TOKEN	Get network settings
UPDATE_NETWORK_TOKEN	Update network settings
Service: CMPart	
ADD_EDGE_SITE_TOKEN	Add edge site
ADD_PARTITION_TOKEN	Add partition settings
ADD_POWER_CIRCUIT_TOKEN	Add power circuit
ADD_RACK_TOKEN	Add rack settings
ADD_SOFTWAREIMAGE_FILE_SELECTION_TOKEN	Add softwareimage file selection
ADD_SOFTWAREIMAGE_TOKEN	Add softwareimage settings
CMDAEMON_CLEAN_STOP_TOKEN	Obtain cleanliness status of stop
CMDAEMON_FAILOVER_SLAVE_RESULT_TOKEN	Obtain status of slave result
CMDAEMON_FAILOVER_SLAVE_TOKEN	Obtain status of slave
CMDAEMON_FAILOVER_STATUS_TOKEN	Obtain status of failover
CMDAEMON_FAILOVER_TOKEN	Set CMDaemon failover condition achieved
CMDAEMON_QUORUM_TOKEN	Set CMDaemon quorum achieved
CMDAEMON_RESOURCE_MIGRATE_TOKEN	Obtain status of resource migration
CMDAEMON_RESOURCE_STATUS_TOKEN	Obtain status of resource
CREATE_RAMDISK_TOKEN	Create ramdisk
EDGE_SITE_SECRET_TOKEN	Show edge site secret
FORGET_RACK_ISOLATION_REQUEST_INFO_TOKEN	Forget rack isolation request info
GET_EDGE_SITE_TOKEN	Get edge site
GET_GNSS_LOCATION_TOKEN	Get GNSS location
GET_NODE_ARCH_OS_TOKEN	Get architecture and OS
GET_PARTITION_TOKEN	Get partition settings
GET_POWER_CIRCUIT_TOKEN	Get power circuit
GET_PRS_STATUS_TOKEN	Get PRS status
GET_RACK_ISOLATION_REQUEST_INFO_TOKEN	Get rack isolation request info
GET_RACK_TOKEN	Get rack settings
GET_RAW_NMX_TOKEN	Get raw NMX settings

...continues

Table J: List Of Tokens...continued

Service and token name	User can...
GET_SOFTWAREIMAGE_FILE_SELECTION_TOKEN	Get softwareimage file selection
GET_SOFTWAREIMAGE_TOKEN	Get softwareimage settings
HANDLE_RACK_ISOLATION_REQUEST_INFO_TOKEN	Handle rack isolation request info
KERNEL_CONFIG_HASH_TOKEN	Kernel config hash
NETQ_SETTINGS_TOKEN	Show NETQ settings
NMXM_SETTINGS_TOKEN	Show NMX Management settings
REQUEST_RACK_ELECTRICAL_ISOLATION_TOKEN	Request rack electrical isolation
REQUEST_RACK_LIQUID_ISOLATION_TOKEN	Request rack liquid isolation
SET_GNSS_LOCATION_TOKEN	Set GNSS location
START_PRS_DOMAIN_TOKEN	Set PRS domain
STOP_PRS_DOMAIN_TOKEN	Stop PRS domain
UFM_SETTINGS_TOKEN	Show UFM settings
UPDATE_EDGE_SITE_TOKEN	Update edge site
UPDATE_GNSS_LOCATION_TOKEN	Update GNSS location
UPDATE_PARTITION_TOKEN	Update partition settings
UPDATE_POWER_CIRCUIT_TOKEN	Update power circuit settings
UPDATE_RACK_TOKEN	Update rack settings
UPDATE_RAW_NMX_TOKEN	Update raw NMX settings
UPDATE_SOFTWAREIMAGE_FILE_SELECTION_TOKEN	Update softwareimage file selection
UPDATE_SOFTWAREIMAGE_TOKEN	Update softwareimage settings
Service: CMProc	
CLEAN_IPC_TOKEN	Clear IPC state
EXEC_COMMAND_TOKEN	Execute a command on a head node
EXEC_INTERNAL_COMMAND_TOKEN	Execute internal command (defined in the source code, RPC API, internal)
GET_ALL_PROCESSES_TOKEN	Retrieve list of all processes that are currently running on a device managed by CMDaemon
GET_MSGQUEUE_TOKEN	Get message queue status
GET_PROCESS_TOKEN	Retrieve list of processes that are currently running on a device managed by CMDaemon
GET_SEMAPHORE_TOKEN	Get semaphore
GET_SHARED_MEM_TOKEN	Get shared memory
SEND_SIGNAL_TOKEN	Send signal to a process
START_SHELL_TOKEN	Start SSH session
Service: CMProv	
ADD_FSPART_TOKEN	Set FSPart (internal)

...continues

Table J: List Of Tokens...continued

Service and token name	User can...
CANCEL_PROVISIONING_REQUEST_TOKEN	Cancel provisioning request
FSPART_BACKUP_TOKEN	Show FSPart backup
GET_FSPART_ASSOCIATION_TOKEN	Get FSPart association
GET_FSPART_TOKEN	Get FSPart (internal)
GET_LAST_PROVISIONING_NODE_TOKEN	Get last provisioning node
GRAB_IMAGEUPDATE_TOKEN	Grab changes from node to software image and vice versa
IMAGEUPDATE_TOKEN	Send image changes to nodes
LOCK_FSPART_TOKEN	Lock FSPart
MANAGE_RSYNC_DAEMON_TOKEN	Manage the rsync process (CMDaemon)
PROVISIONERS_STATUS_TOKEN	Check status of provisioners e.g. images are in sync
REQUEST_PROVISIONING_TOKEN	Request provisioning (nodes with a provisioning role)
RUN_FSPART_SYNC_SCRIPT_TOKEN	RUN FSPart sync script
RUN_PROVISIONINGPROCESSORJOB_ TOKEN	Start and run a provisioning job (nodes with a provisioning role)
UPDATEPROVISIONERS_TOKEN	Synchronize software images across provisioning systems (requires at least two provisioners)
UPDATE_CONFIG_FILES_AFTER_IMAGE_ UPDATE_TOKEN	Update config files after image update
UPDATE_FSPART_ASSOCIATION_TOKEN	Update FSPart association
UPDATE_FSPART_TOKEN	Update FSPart (internal)
UPDATE_PROVISIONINGPROCESSORJOB_ TOKEN	Update status of running provisioning jobs (CMDaemon)
Service: CMServ	
CALLINIT_OSSERVICE_TOKEN	Call init (useful for the node-installer itself)
GET_OSSERVICE_TOKEN	Get system service information
RELOAD_OSSERVICE_TOKEN	Reload system services
RESET_OSSERVICE_TOKEN	Reset system services
RESTART_OSSERVICE_TOKEN	Restart system services
RESTART_WLM_OSSERVICE_TOKEN	Restart WLM systemservices
START_OSSERVICE_TOKEN	Start system services (service foo start)
STOP_OSSERVICE_TOKEN	Stop system services
UPDATE_OSSERVICE_TOKEN	Update system services
Service: CMSession	
CLIENT_USER_DATA_TOKEN	Show client user data
END_SESSION_TOKEN	Terminate sessions
GET_BROADCAST_EVENTS_TOKEN	Receive broadcast events
GET_SESSION_TOKEN	Retrieve session information
HANDLE_EVENT_TOKEN	Handle events

...continues

Table J: List Of Tokens...continued

Service and token name	User can...
LIST_CLIENT_USER_DATA_TOKEN	List client user data
REGISTER_LITENODE_SESSION_TOKEN	Register lite node
REGISTER_NODE_SESSION_TOKEN	Register new nodes in a special CMDaemon session (node-installer)
Service: CMStatus	
GET_DEVICE_STATUS_TOKEN	Get device status
INTERNAL_STATUS_TOKEN	Show internal status information
SET_DEVICE_STATUS_TOKEN	Set device status
STATUS_INFO_TOKEN	Show status information
STATUS_MANAGE_TOKEN	Show managed status
Service: CMTest	
GET_MANAGERS_STATE_TOKEN	Get managers state
Service: CMUser	
ADD_GROUP_TOKEN	Add a new LDAP group
ADD_USER_TOKEN	Add a new LDAP user
CHECK_ACCESS_TOKEN	Check project manager access
CREATE_MISSING_HOME_DIRECTORIES_TOKEN	Create missing home directories
GET_DISABLED_PASSWORD_SSH_TOKEN	Get disabled ssh password
GET_GROUP_TOKEN	Retrieve group information
GET_USER_TOKEN	Retrieve user information
REGENERATE_USER_CERTIFICATES_TOKEN	Regenerate user certificates
SET_DISABLED_PASSWORD_SSH_TOKEN	Disable password for SSH
SET_USER_CLOUD_JOB_TOKEN	Set user cloud job
SET_USER_PROFILE__TOKEN	Set user profile
UPDATE_GROUP_TOKEN	Modify an existing LDAP group
UPDATE_USER_TOKEN	Modify an existing LDAP user

K

Understanding Consolidation

K.1 Introduction

Consolidation is discussed in the sections on using consolidation in the Monitoring chapter (sections 10.4.3 and 10.5.2).

However, it may be confusing to have the concept of consolidation discussed in the same place as the use of consolidation. Also, the algebra that appears in that discussion (page 595) may not appeal to people. There are many who would like an explanation that may be more intuitive, even if it is less rigorous.

Therefore, in this section a more informal and visual approach is taken to explain consolidation.

K.2 What Is Consolidation?

Consolidation is the compression of data, for data values that have been measured over a fixed interval.

The compression is nothing particularly sophisticated. It is carried out by using some simple mathematical functions to the data points: the average, the maximum, or the minimum.

K.3 Raw Data And Consolidation

Suppose raw data is sampled every 2 minutes.

And the raw data values are consolidated every 10 minutes.

A visual representation of the data values available to the system is:

```
                        --- time --->
raw:            | | | | | | | | | | | | | | | | | | | |
consolidated:   |       |       |       |       |
```

Here, every "|"" indicates a data point, so that the visual shows 5 times as many raw data points as consolidated data values.

In the preceding visual it makes no sense to use consolidated data since the data values for raw data and consolidated data overlap. I.e., the more accurate raw data values exist for the entire period.

As time passes, the intention is to start dropping old raw data, to save space on the disk.

For example, for the first 20 minutes in the following visual, there are no longer raw data values available:

Example

```
                        --- time --->
raw:                          | | | | | | | | | | | | | | |
consolidated:   |       |       |       |       |       |
```

But the consolidated data points for this period are still available to the system.

When the data values are plotted in Base View graphs, periods without raw data values automatically have consolidated data values used.

So a combination of both data sources is used, which can be visually represented with:

Example

```
                          --- time --->
plot:           |       |       | | | | | | | | | | | | | |
```

That behavior holds true for `cmsh` too.

The behavior illustrated in the last visual assumes that the cluster has been UP for long enough that raw data is being dropped.

In this case, "long enough" means at least 7 days.

However, because RLE (Run Length Encoding) is used to compress the sampled monitoring data values on disk, this minimal "long enough" time can be (much) longer than 7 days. It depends on how much the measurable that is being sampled is changing as each sample is taken. if it is not changing, then RLE can compress over a longer time period.

For example, if a node has been up and reachable without issues for 1000 days, then the `ssh2node` health check raw data values would be `PASS` over that 1000 days. For the period from now to 7 days ago, the raw data values of `PASS` are kept as they are for now. However, for the period from 7 days ago to 1000 days ago, consolidation on the unchanging raw values means that only two values, namely the `PASS` value of 1000 days and the `PASS` value of 7 days ago, need to be retained, in order to have a totally accurate record of what the values were in that period.

On the other hand, the `forks` metric changes very quickly, and thus can do little RLE compression. That makes it a good choice for demonstrating the kind of output that the preceding visuals imply.

K.4 A Demonstration Of The Output

So, as a demonstration, the last 7 days for forks are now shown, with the data values in the middle elided:

Example

```
[basecm11->device[basecm11]]% dumpmonitoringdata -7d now forks
Timestamp                  Value                Info
-------------------------- -------------------- ----------
2018/10/17 10:30:00        2.76243 processes/s
2018/10/17 11:30:00        2.52528 processes/s
2018/10/17 12:30:00        2.53972 processes/s
...
2018/10/24 10:42:00        2.66669 processes/s
2018/10/24 10:44:00        2.63333 processes/s
2018/10/24 10:46:00        2.64167 processes/s
```

The first part of the output shows samples listed every hour. These are the consolidated data values.

The last part of the output shows samples listed every 2 minutes. These are the raw data value values.

I.e.: consolidated data values are used beyond a certain time in the past.

If the administrator would like to explore this further, then displaying only consolidation values is possible in `cmsh` by using the `--consolidationinterval` option of the `dumpmonitoringdata` command:

Example

```
[basecm11->device[basecm11]]% dumpmonitoringdata --consolidationinterval 1h -7d now forks
Timestamp                  Value                Info
-------------------------- -------------------- ----------
2019/01/07 11:39:06        2.65704 processes/s
2019/01/07 12:30:00        2.60111 processes/s
2019/01/07 13:30:00        2.58328 processes/s
...
[basecm11->device[basecm11]]% dumpmonitoringdata --consolidationinterval 1d -7d now forks
Timestamp                  Value                Info
-------------------------- -------------------- ----------
2019/01/07 18:09:06        2.59586 processes/s
2019/01/08 13:00:00        2.58953 processes/s
2019/01/09 06:06:06        2.58854 processes/s
[basecm11->device[basecm11]]% dumpmonitoringdata --consolidationinterval 1w -7d now forks
Timestamp                  Value                Info
-------------------------- -------------------- ----------
2019/01/08 11:15:12.194    2.59113 processes/s
```

L

Node Execution Filters And Execution Multiplexers

Node execution filters and execution multiplexers define where data producers are executed on the nodes of a cluster, and what nodes are targeted to obtain the data.

This appendix explains how node execution filters and execution multiplexers work with the help of some explicit basic examples. The aim is to have the cluster administrator understand how they work and how to use them.

The reference cluster in this section is a 5-node cluster, made up of a head node (`basecm11`) and 4 regular nodes (`node001..node004`). The commands run in this appendix are carried out during a `cmsh` session that continues on from the point that it left off earlier.

The terms "node execution filters" and "execution multiplexers" are commonly abbreviated to filters and multiplexers in this appendix.

A simple custom data producer script is created and used to explain some of the more-involved concepts of filters and multiplexers more clearly. The custom script is:

```
[root@basecm11 ~]# cat /cm/shared/fm.sh
#!/bin/bash

echo $((RANDOM%100))
echo "Sampled on $(hostname) for $CMD_HOSTNAME" >&3
```

It should be made executable, for example, with `chmod a+x`. When run, the script outputs a random number, it outputs the host it is being run on (`$(hostname)`), and also the host the metric is targeting (`$CMD_HOSTNAME`). The hosts that it is run on can be defined by filters, while the hosts that are targeted by the metric can be defined by multiplexers.

The term multiplex: The word "multiplex" can be confusing to system administrators. In electronics, the term multiplex implies that signals are being gathered from various inputs, and going into a main input.

Here the idea is applicable to the signals (samples) from the execution multiplexers (nodes where the samples are). The samples are multiplexed (gathered) from those nodes, to the node (or nodes) where the data producer is executing.

- The execution of the data producer is on the node (or nodes) defined by `nodeexecutionfilter`. The data producer execution nodes are the ones listed using the `nodes` command of `cmsh`.
- The nodes where the samples are obtained from are defined by the `executionmultiplexer` setting. Those muliplexer nodes can have their samples displayed as output using the `samplenow` command (section 10.6.2, page 628) of `cmsh`.

L.1 Data Producers: Default Configuration For Running And Sampling

If there is no configuration defined for the data producer in the filters or multiplexers for that data producer, then each node runs a data producer on itself, and that data producer targets the node that it is running on.

For example, the existing `dmesg` data producer comprises the `dmesg` health check (section G.2.1) and by default has no filter or multiplexer defined for it. If an attempt is made to list any filter or multiplexer for `dmesg`, then by default there is no content under the table headings:

```
[root@basecm11 ~]# cmsh
[basecm11]% monitoring setup
[basecm11->monitoring->setup]% nodeexecutionfilters dmesg; list; ..;..
Type         Name (key)               Filter                   Filter operation
------------ ------------------------ ------------------------ ----------------
[basecm11->monitoring->setup]% executionmultiplexers dmesg; list; ..;..
Name (key)
------------------------
```

Another way of seeing that no such filters or multiplexers have been defined for `dmesg` could be by seeing that none are defined in its submodes:

```
[basecm11->monitoring->setup]% show dmesg | grep submode
Execution multiplexer            <0 in submode>
Node execution filters           <0 in submode>
```

Most existing data producers have filters and multiplexers defined. The number of filters and multiplexers set per data producer can conveniently be viewed via list formatting:

Example

```
[basecm11->monitoring->setup]% list -f name,nodeexecutionfilters,executionmultiplexer | more
name (key)           nodeexecutionfilters executionmultiplexer
-------------------- -------------------- --------------------
AggregateNode        <1 in submode>       <1 in submode>
AggregatePDU         <1 in submode>       <1 in submode>
AlertLevel           <1 in submode>       <1 in submode>
CMDaemonState        <0 in submode>       <0 in submode>
Cassandra            <1 in submode>       <0 in submode>
ClusterTotal         <1 in submode>       <0 in submode>
DeviceState          <1 in submode>       <0 in submode>
...
```

L.1.1 Nodes That Data Producers Are Running On By Default—The `nodes` Command

The `nodes` command shows which nodes the data producer runs on. By default, the data producer runs on all nodes, when nothing has been set explicitly, because each node runs the data producer for itself:

```
[basecm11->monitoring->setup]% nodes dmesg
node001..node004,basecm11
```

L.1.2 Nodes That Data Producers Target By Default—The `samplenow` Command

Nodes where samples are being obtained at can be seen using the `samplenow` command for the specified nodes.

Again, by default, each node is a target, because the target is same node that the `dmesg` data producer runs on:

```
[basecm11->monitoring->setup]% device samplenow -t node dmesg
Entity       Measurable    Type         Value      Age          Info
------------ ------------  ------------ ---------- ----------  ----------
node001      dmesg         OS           PASS       0.093s
node002      dmesg         OS           PASS       0.087s
node003      dmesg         OS           PASS       0.088s
node004      dmesg         OS           PASS       0.09s
basecm11     dmesg         OS           PASS       0.179s
```

In the outputs to `samplenow` displayed in this appendix, some columns are omitted for the sake of clarity.

The `-t node` option to `samplenow` expands to `-n node001..node004,basecm11` for this reference cluster.

L.2 Data Producers: Configuration For Running And Targeting

Filters and multiplexers define which nodes run the data producers, and which nodes are the targets for measurables.

The `fm.sh` script introduced on page 1073 can be used to define several custom metrics according to what nodes run the script and what nodes are targeted by the script.

L.2.1 Custom Metrics From The `fm.sh` Custom Script

Custom data producers of type `metric` are created in this section. These data producers comprise custom metrics, which are now set up with varying filtering and multiplexing definitions, to illustrate how the definitions work.

The Metric `all_for_self`

The metric from the data producer `all_for_self` can be set up with no filtering or multiplexing defined, as follows:

```
[basecm11->monitoring->setup]% add metric all_for_self
[basecm11->monitoring->setup*[all_for_self*]]% set consolidator none
[basecm11->monitoring->setup*[all_for_self*]]% set script /cm/shared/fm.sh
[basecm11->monitoring->setup*[all_for_self*]]% set class Test
[basecm11->monitoring->setup*[all_for_self*]]% commit
```

The value for `class` is mandatory but arbitrary. It is an arbitrary grouping mechanism, which can be useful in Base View for grouping folders in trees.

Sampling results for the metric `all_for_self`: With no filtering or multiplexing, the metric just runs everywhere by default, with the target of the running metric being itself too. The `Info` field output from `samplenow` shows this behavior in the script output:

```
[basecm11->monitoring->setup[all_for_self]]% exit
[basecm11->monitoring->setup]% device samplenow -t node all_for_self
Entity       Measurable    Type  Value  Age     Info
------------ ------------- ----- ------ ------  ----------------------------------
node001      all_for_self  Test  2      0.08s   Sampled on node001 for node001
node002      all_for_self  Test  13     0.084s  Sampled on node002 for node002
node003      all_for_self  Test  97     0.08s   Sampled on node003 for node003
node004      all_for_self  Test  18     0.084s  Sampled on node004 for node004
basecm11     all_for_self  Test  63     0.144s  Sampled on basecm11 for basecm11
```

The Metric `some_for_self`

The metric `some_for_self` can be set up with filtering set up for some nodes, and no multiplexing set up, as follows:

```
# on node001,node002 for itself
[basecm11->monitoring->setup]% add metric some_for_self
[basecm11->monitoring->setup*[some_for_self*]]% set consolidator none
[basecm11->monitoring->setup*[some_for_self*]]% set class Test
[basecm11->monitoring->setup*[some_for_self*]]% set script /cm/shared/fm.sh
[basecm11->...*[some_for_self*]]% nodeexecutionfilters
[basecm11->...*[some_for_self*]->nodeexecutionfilters]% add node some_nodes
[basecm11->...->nodeexecutionfilters*[some_nodes*]]% set nodes node001 node002
[basecm11->...->nodeexecutionfilters*[some_nodes*]]% show
Parameter                      Value
------------------------------ ------------------------------------------------
Filter operation               Include
Name                           some_nodes
Nodes                          node001,node002
Revision
Type                           Node
[basecm11->...->nodeexecutionfilters*[some_nodes*]]% commit
```

Sampling results for the metric `some_for_self`**:** Only some filtering defined, and no multiplexing defined at all, means that the metric just runs on the filtered nodes, and targets only the nodes defined in filter too:

```
[basecm11->...->nodeexecutionfilters[some_nodes]]% ..; ..; ..
[basecm11->monitoring->setup]% device samplenow -t node some_for_self
Entity       Measurable     Type  Value  Age    Info
------------ -------------- ----- ------ ------ ------------------------------
node001      some_for_self  Test  88     0.094s Sampled on node001 for node001
node002      some_for_self  Test  98     0.075s Sampled on node002 for node002
```

The Metric `from head_for_some_others`

The metric `from_head_for_some_others` can be set up with filtering defined for the head node, and multiplexing defined for some other regular nodes (other than `node001` and `node002` here), as follows:

```
# on active head for node003,node004
[basecm11->monitoring->setup]% add metric from_head_for_some_others
[basecm11->monitoring->setup*[from_head_for_some_others*]]% set consolidator none
[basecm11->monitoring->setup*[from_head_for_some_others*]]% set script /cm/shared/fm.sh
[basecm11->monitoring->setup*[from_head_for_some_others*]]% set class Test
[basecm11->...*[from_head_for_some_others*]]% nodeexecutionfilters
[basecm11->...*[from_head_for_some_others*]->nodeexecutionfilters]% active
Added active resource filter
[basecm11->...*[from_head_for_some_others*]->nodeexecutionfilters]% show active head node
Parameter                      Value
------------------------------ ------------------------------------------------
Filter operation               Include
Name                           Active head node
Operator                       OR
Resources                      Active
Revision
Type                           Resource
[basecm11->monitoring->setup*[from_head_for_some_others*]->nodeexecutionfilters]% ..
[basecm11->...*[from_head_for_some_others*]]% executionmultiplexers
```

```
[basecm11->...*[from_head_for_some_others*]->executionmultiplexers]% add node other_nodes
[basecm11->..._for_some_others*]->executionmultiplexers*[other_nodes*]]% set nodes node003 node004
[basecm11->...*[from_head_for_some_others*]->executionmultiplexers*[other_nodes*]]% show
Parameter                      Value
------------------------------ ------------------------------------------------
Filter operation               Include
Name                           other_nodes
Nodes                          node003,node004
Revision
Type                           Node
[basecm11->..._for_some_others*]->executionmultiplexers*[other_nodes*]]% commit; ..; ..;..
[basecm11->monitoring->setup]%
```

The filter is given a resource, `Active`, which is a way to set the filter for the active node only. Available resources for a node can be seen by running the command `monitoringresources` for a device:

Example

```
[basecm11->monitoring->setup]% device monitoringresources basecm11
Active
Ethernet
RDO
backup
boot
...
```

Sampling results for the metric `from head_for_some_others`**:** With filtering defined for the active head node, and multiplexing defined for those other nodes, it means that the metric targets those other nodes, and the metric runs on the head node. That is, the other nodes are targeted by the head node that is running the metric:

```
[basecm11->monitoring->setup]% device samplenow -t node from_head_for_some_others
Entity       Measurable               Type  Value Age    Info
------------ ------------------------ ----- ----- -----  --------------------------------
node003      from_head_for_some_others  Test 20    0.084s Sampled on basecm11 for node003
node004      from_head_for_some_others  Test 44    0.067s Sampled on basecm11 for node004
```

The `nodes` command confirms that the head node, `basecm11` is the filtered node, that is, the only node(s) running the metric:

```
[basecm11->monitoring->setup]% nodes from_head_for_some_others
basecm11
```

L.3 Replacing A Resource With An Explicit Node Specification

Within the filter `Active head node`, associated with the metric `from_head_for_some_others`, the resource object `Active` can be replaced with a `node` object instead, if the node is defined as being `basecm11`. Doing this on a high-availability cluster where there is one active and one passive head node, would be unwise. However, doing this on the reference cluster for teaching purposes is of course absolutely fine because it helps make things a bit more concrete for the reader. The replacement can be carried out for the session as follows:

```
[basecm11->monitoring->setup]% nodeexecutionfilters from_head_for_some_others
[basecm11->...[from_head_for_some_others]->nodeexecutionfilters]% list
Type         Name (key)              Filter                   Filter operation
------------ ----------------------- ------------------------ ----------------
```

```
Resource     Active head node         Active                    Include
[basecm11->...[from_head_for_some_others]->nodeexecutionfilters]% remove active head node
[basecm11->...*[from_head_for_some_others*]->nodeexecutionfilters*]% add node head
[basecm11->...*[from_head_for_some_others*]->nodeexecutionfilters*[head*]]% set nodes basecm11
[basecm11->...*[from_head_for_some_others*]->nodeexecutionfilters*[head*]]% show
Parameter                      Value
------------------------------ ------------------------------------------------
Filter operation               Include
Name                           head
Nodes                          basecm11
Revision
Type                           Node
[basecm11->...*[from_head_for_some_others*]->nodeexecutionfilters*[head*]]% commit
[basecm11->...[from_head_for_some_others]->nodeexecutionfilters*[head]]% ..;..;..
```

The sample results are the same kind of output, and the filter node used is the same. The target sample outputs are the same kind of output as before the replacement. The filter node on which the metric runs is also seen to be the same:

```
[basecm11->monitoring->setup]% device samplenow -t node from_head_for_some_others
Entity       Measurable                 Type  Value Age    Info
------------ -------------------------- ----- ----- ------ ----------------------------------
node003      from_head_for_some_others  Test  11    0.081s Sampled on basecm11 for node003
node004      from_head_for_some_others  Test  12    0.07s  Sampled on basecm11 for node004
[basecm11->monitoring->setup]% nodes
basecm11
```

L.4 Excessive Sampling

If another node is appended to the node in the filter, then a warning comes up.

```
[basecm11->monitoring->setup]% nodeexecutionfilters from_head_for_some_others
[basecm11->...[from_head_for_some_others]->nodeexecutionfilters]% append head nodes node003
[basecm11->...*[from_head_for_some_others*]->nodeexecutionfilters*]% commit
========================= from_head_for_some_others ==========================
Field                   Message
----------------------- ------------------------------------------------------
executionMultiplexers   Warning: Execution filters/multiplexers are set to run
                        on multiple nodes for the same target. This likely
                        means they are badly configured


[basecm11->...[from_head_for_some_others]->nodeexecutionfilters]% show head

Parameter               Value
----------------------- ------------------------------------------------------
Filter operation        Include
Name                    head
Nodes                   node003,basecm11
Revision
Type                    Node
```

The warning is there because the node execution filters are doing the same thing from different nodes for an execution multiplexer target, and duplicating monitoring execution is typically a waste of resources, and thus typically a mistake.

However, the warnings are merely warnings, and not errors. So BCM just goes ahead with setting up the filter/muliplex system according to what the administrator has specified. The `nodes` and `samplingnow` commands now show:

```
[basecm11->...[from_head_for_some_others]->nodeexecutionfilters]% ..;..
[basecm11->monitoring->setup]% device samplenow -t node from_head_for_some_others
Entity       Measurable                Type  Value Age     Info
------------ ------------------------- ----- ----- ------  --------------------------------
node003      from_head_for_some_others  Test  3     0.151s  Sampled on basecm11 for node003
node003      from_head_for_some_others  Test  76    0.08s   Sampled on node003 for node003
node004      from_head_for_some_others  Test  25    0.08s   Sampled on node003 for node004
node004      from_head_for_some_others  Test  88    0.151s  Sampled on basecm11 for node004
[basecm11->monitoring->setup]% nodes from_head_for_some_others
node003,basecm11
```

The node `node003` is now doing what the head node is, sampling the same targets, which is typically a bad idea. However the behavior is indeed as expected for this particular configuration.

Whether running a particular configuration is actually wise, is up to the administrator—but in any case the filter/multiplex system allows plenty of abuse of this kind.

L.5 Not Just For Nodes

Nodes are what node execution filters and execution multiplexers run on. However, sometimes it is more convenient to execute based on other types.

The possible types can be listed with tab-completion suggestions when adding a node execution filter or an execution multiplier:

```
[basecm11->monitoring->setup[dmesg]->nodeexecutionfilters]% add<TAB><TAB>
category  lua       node      overlay   resource  type
```

L.6 Lua Node Execution Filters

Lua (`https://www.lua.org/`) is a lightweight scripting language embedded into CMDaemon. It allows more advanced node execution filters to be written using a Lua filter file.

Example

```
[basecm11->monitoring->setup[<data producer>]->nodeexecutionfilters]% add lua lua-filter
[basecm11->...nodeexecutionfilters*[lua-filter*]]% set code <Lua filter file name>
```

In the preceding example session, the name `lua-filter` is an arbitrary name, that is added to the object that is associated with the Lua filter file *<Lua filter file name>*.

The `self` Lua table is passed by CMDaemon, and contains the entity for which the filter is evaluated. Only devices have a `self` table that is not `nil`. All other entities have a `self` table that is `nil`.

For example, a filter can be created based on a regex match of the hostname:

Example

```
if self == nil then
  return false
else
  return self.hostname:match("^node[0-9]+") ~= nil)
end
```

The Lua filter is evaluated for all nodes:

Example

```
[basecm11->monitoring->setup[dmesg]]% nodes
node001..node004
```

Development of Lua filters is best done outside of CMDaemon. Doing so requires the cluster administrator to create node environments to be evaluated by hand:

Example

```
[root@basecm11 ~]# cat node001.lua
self = {}
self.hostname = "node001"
self.category = "default"

[root@basecm11 ~]# cat basecm11.lua
self = {}
self.hostname = "basecm11"
```

This is in addition to the original filter script:

Example

```
[root@basecm11 ~]# cat filter.lua
if self == nil then
  return false
else
  return self.hostname:match("^node[0-9]+") ~= nil
end
```

Both nodes can then be run through the filter using the `lua` interpreter:

Example

```
[root@basecm11 ~]# lua
Lua 5.1.4  Copyright (C) 1994-2015 Lua.org, PUC-Rio
> dofile('basecm11.lua')
> print(dofile('filter.lua'))
false
> dofile('node001.lua')
> print(dofile('filter.lua'))
true
```

The Lua `self` exported by CMDaemon contains the following:

- `self`: The main table object, `nil` for non-devices.
- `self.hostname`: The hostname of the device.
- `self.partition`: The name of the partition to which the device belongs.
- `self.category`: The name of the category to which the compute node belongs.
- `self.nodegroups`: An array with the name of node groups the node belongs to.
- `self.mac`: The MAC address of the device.
- `self.ip`: The IP of the device, only available for non-node devices.
- `self.network`: The IP address of the device, only available for non-node devices.
- `self.interfaces`: The array of interface of a node.

- `self.interfaces[1].name`: The name of the first interface.
- `self.interfaces[1].ip`: The IP address of the first interface.
- `self.interfaces[1].network`: The name of the network of the first interface.
- `self.roles`: The array of roles of a node.
- `self.roles[1].name`: The name of the first role.
- `self.roles[1].type`: The type of the first role.
- `self.status.status`: The status of the device.
- `self.status.user_message`: The user message set for the device.
- `self.status.info_message`: The information message set for the device.
- `self.status.closed`: A boolean marking the node as closed.
- `self.status.restart_required`: A boolean marking the node needs to be rebooted.
- `self.status.healthcheck_failed`: A boolean marking at least one health check has returned `FAIL`.
- `self.status.healthcheck_unknown`: A boolean marking at least one health check has returned `UNKNOWN`.
- `self.status.state_flapping`: A boolean marking the node status transitioning often in a short time span
- `self.system.name`: The name of the system.
- `self.system.manufacturer`: The manufacturer of the system.
- `self.system.motherboard.name`: The name of the motherboard.
- `self.system.motherboard.manufacturer`: The manufacturer of the motherboard.
- `self.system.bios.version`: The BIOS version.
- `self.system.bios.vendor`: The BIOS vendor name.
- `self.system.bios.date`: The BIOS date.
- `self.system.os.name`: The OS name.
- `self.system.os.version`: The OS version.
- `self.system.os.flavor`: The OS flavor.

Using the `self` environment, complex filters can easily be created. For example, the following filter can be built to include only the nodes on the IB network, which also have a Slurm client role:

Example

```
[root@basecm11 ~]# cat ib-slurm-filter.lua
if self == nil then
  return false
end

on_ib_network = false
```

```
for index, interface in ipairs(self.interfaces) do
  on_ib_network = on_ib_network or (interface.network == "ibnet")
done

slurm_client = false
for index, role in ipair(self.roles) do
  slurm_client = slurm_client or (role.name == "SlurmClient")
done

return on_ib_network and slurm_client
```

It is also possible to use external sources, like the file system, to determine the filter for a node.

Example

```
[root@basecm11 ~]# cat file-check-filter.lua
if self == nil then
  return false
end

function file_exists(name)
  local f = io.open(name, "r")
  if f ~= nil then
    io.close(f)
    return true
  else
    return false
  end
end

return file_exists(string.format('/opt/filter/%s', self.hostname))
```

It is important to understand that this Lua script is evaluated for all nodes, on the active head node. The Lua script should therefore be fast, and return within a few milliseconds.

M

A Tree View Of cmsh

M.1 Modes

A 3-level tree of the modes in cmsh is:

```
|-- category
|   |-- biossettings
|   |-- bmcsettings
|   |-- dpusettings
|   |   `-- keyvaluesettings
|   |-- fsexports
|   |-- fsmounts
|   |-- gpusettings
|   |-- kernelmodules
|   |-- roles
|   |   |-- advancedsettings
|   |   |-- commsettings
|   |   |-- configs
|   |   |-- configurations
|   |   |-- connectionsettings
|   |   |-- domains
|   |   |-- engines
|   |   |-- environments
|   |   |-- excludelistsnippets
|   |   |-- genericresources
|   |   |-- interfaces
|   |   |-- logsettings
|   |   |-- momsettings
|   |   |-- nginxreverseproxy
|   |   |-- nodecustomizations
|   |   |-- openports
|   |   |-- policies
|   |   |-- powerprofiles
|   |   |-- resourceproviders
|   |   |-- routes
|   |   |-- servers
|   |   |-- spawner
|   |   |-- storagedrivers
|   |   |-- storagebackends
|   |   `-- zones
|   |-- selinuxsettings
|   |   `-- keyvaluesettings
|   |-- services
```

```
|   |-- staticroutes
|   `-- ztpsettings
|-- cert
|-- cloud
|   |-- extensions
|   |-- instancepools
|   |-- ocigpumemoryclusters
|   |-- regions
|   |-- types
|   `-- vpcs
|-- configurationoverlay
|   |-- customizations
|   `-- roles
|       |-- advancedsettings
|       |-- commsettings
|       |-- configs
|       |-- configurations
|       |-- connectionsettings
|       |-- domains
|       |-- engines
|       |-- environments
|       |-- excludelistsnippets
|       |-- genericresources
|       |-- interfaces
|       |-- logsettings
|       |-- momsettings
|       |-- nginxreverseproxy
|       |-- nodecustomizations
|       |-- openports
|       |-- policies
|       |-- powerprofiles
|       |-- resourceproviders
|       |-- routes
|       |-- servers
|       |-- spawner
|       |-- storagedrivers
|       |-- storagebackends
|       `-- zones
|-- device
|   |-- accesssettings
|   |-- biosettings
|   |-- bmcsettings
|   |-- chassisposition
|   |-- cloudsettings
|   |   |-- disks
|   |   |-- platformconfig
|   |   `-- storage
|   |-- dpusettings
|   |   `-- keyvaluesettings
|   |-- fsexports
|   |-- fsmounts
|   |-- gpusettings
|   |-- interfaces
|   |-- kernelmodules
|   |-- nvconfiguration
```

```
|   |-- prometheusmetricforwarders
|   |-- proxysettings
|   |-- rackposition
|   |-- roles
|   |   |-- advancedsettings
|   |   |-- commsettings
|   |   |-- configs
|   |   |-- configurations
|   |   |-- connectionsettings
|   |   |-- domains
|   |   |-- engines
|   |   |-- environments
|   |   |-- excludelistsnippets
|   |   |-- genericresources
|   |   |-- interfaces
|   |   |-- logsettings
|   |   |-- momsettings
|   |   |-- nginxreverseproxy
|   |   |-- nodecustomizations
|   |   |-- openports
|   |   |-- policies
|   |   |-- powerprofiles
|   |   |-- resourceproviders
|   |   |-- routes
|   |   |-- servers
|   |   |-- spawner
|   |   |-- storagedrivers
|   |   |-- storagebackends
|   |   `-- zones
|   |-- selinuxsettings
|   |   `-- keyvaluesettings
|   |-- services
|   |-- snmpsettings
|   |-- staticroutes
|   `-- ztpsettings
|       `-- keyvaluesettings
|-- edgesite
|-- etcd
|-- fspart
|   `-- excludelistsnippets
|-- group
|-- hierarchy
|   |-- sources
|   `-- targets
|-- kubernetes
|   |-- appgroups
|   |   `-- applications
|   |-- labelsets
|   `-- users
|-- main
|-- monitoring
|   |-- action
|   |-- consolidator
|   |   `-- consolidators
|   |-- labeledentity
```

```
|   |-- measurable
|   |-- query
|   |   `-- drilldown
|   |-- report
|   |-- setup
|   |   |-- dpusettings
|   |   |-- executionmultiplexers
|   |   |-- jobmetricsettings
|   |   `-- nodeexecutionfilters
|   |-- standalone
|   `-- trigger
|       `-- expression
|-- network
|-- nodegroup
|-- partition
|   |-- accesssettings
|   |-- archos
|   |-- bmcsettings
|   |-- burnconfigs
|   |-- dpusettings
|   |   `-- keyvaluesettings
|   |-- failover
|   |-- failovergroups
|   |-- leakactionpolicies
|   |   `-- rules
|   |-- netqsettings
|   |   `-- prometheusmetricforwarders
|   |-- nmxmsettings
|   |   `-- prometheusmetricforwarders
|   |-- prometheusmetricforwarders
|   |-- provisioningsettings
|   |-- resourcepools
|   |-- selinuxsettings
|   |   `-- keyvaluesettings
|   |-- snmpsettings
|   |-- ufmsettings
|   |   `-- prometheusmetricforwarders
|   |-- wlmjobpowerusagesettings
|   |-- ztpnewswitchsettings
|   |   `-- keyvaluesettings
|   `-- ztpsettings
|       `-- keyvaluesettings
|-- powercircuit
|-- process
|-- profile
|-- rack
|-- session
|-- softwareimage
|   |-- kernelmodules
|   `-- selection
|-- task
|-- user
|   `-- projectmanager
`-- wlm
    |-- accounting
```

```
|-- cgroups
|-- chargeback
|-- jobqueue
|-- jobs
|-- licenses
|-- ocisettings
|-- placeholders
|-- prssettings
`-- topologysettings
    |-- blocksettings
    |-- parameters
    |-- topographsettings
    `-- treesettings
```

N

Redfish Management

This appendix gives an overview of Redfish management for cluster devices. Redfish is a standard RESTful API for managing servers, storage, and network infrastructure. BCM provides integration with Redfish-enabled devices for out-of-band management, hardware monitoring, and configuration.

Section N.1 provides an overview of Redfish and its role in cluster management.

Section N.2 covers commands for querying and managing devices using Redfish.

N.1 Redfish Overview

Redfish is an industry-standard specification developed by the DMTF (Distributed Management Task Force) that provides a RESTful interface for managing modern data center hardware. It replaces older protocols like IPMI with a more modern, secure, and extensible approach.

N.1.1 Redfish Features

Redfish features include:

- RESTful API using JSON data format
- Secure communication over HTTPS
- Standard schema for hardware components (`https://redfish.dmtf.org/redfish/schema_index`)
- Event-driven notifications
- Support for firmware updates
- Power and thermal management

N.1.2 Redfish In BCM

BCM uses Redfish for:

- BMC (Baseboard Management Controller) communication (section 3.7)
- Hardware inventory collection (section 14.11.1)
- Power control operations (section N.2.3)
- Sensor and health monitoring (section G.2.3 and G.1.10)
- Firmware management (section 14.5.2)
- Leak detection in NVL8 hardware (section 3.3.2 of the *NVIDIA Mission Control Manual*)

N.2 Redfish Commands

Redfish operations can be performed using commands available in the `device` mode of `cmsh`.

N.2.1 Redfish Status

Device status and Redfish connectivity can be verified through standard `device status` (page 291) commands, just as for nodes:

Example

```
[basecm11->device[b13-1-s1-ibcl-01]->interfaces[rf0]]% show
Parameter                      Value
------------------------------ ------------------------------------------------
Revision
Type                           bmc
Network device name            rf0
Network                        ipminet12
...
[basecm11->device[b13-1-s1-ibcl-01]->interfaces[rf0]]% ds b13-1-s1-ibcl-01
b13-1-s1-ibcl-01 ......... [   UP   ]
```

N.2.2 Hardware Inventory Using Redfish

The `hardwareinventoryinfo` command (section 14.11.1) retrieves hardware inventory data from devices using Redfish when available. This includes component information such as CPUs, memory, GPUs, and network adapters.

N.2.3 Power Control Using Redfish

Power control operations (section 4.2) can be performed on Redfish-enabled devices using the standard `power` commands in device mode:

Example

```
[basecm11->device]% power on node001
[basecm11->device]% power off node001
[basecm11->device]% power reset node001
```

For devices with Redfish BMCs, these commands are associated with the `DC power cycle` terminology used in Redfish. The commands communicate with the BMC using the Redfish API to perform the requested power action and assume that the AC power source to the BMC is on.

`AC power cycle` (page 761) is the term used to describe a power reset of the BMC itself.

N.2.4 Redfish Subscriptions: `redfishsubscriptions`

The `redfishsubscriptions` command is run from `device` mode, and displays the status of Redfish event subscriptions for devices such NVLink switches.

BCM subscribes to Redfish events from device BMCs to receive real-time notifications about hardware events, alerts, and status changes.

Example

```
[basecm11->device]% redfishsubscriptions
hostname            id          initialized  ip              kind     port  ssl   subscribed  uuid
------------------- ----------- ------------ --------------- -------- ----- ----- ----------  --------...--
a07-p01-dgx-02-c01  3111116886  true         10.102.174.132  GB300    443   true  true        1be5d...97
a07-p01-dgx-02-c02  454918174   true         10.102.174.133  GB300    443   true  true        a8eb0...b4
a07-p01-dgx-02-c03  2737332992  true         10.102.174.134  GB300    443   true  true        8754c...79
...
```

```
a07-p01-nvsw-01     2220290718  true         10.102.174.150  GB200SW  443   true  true        58519...65
a07-p01-nvsw-02     2506796990  true         10.102.174.151  GB200SW  443   true  true        1f792...61
...
```

The output shows:

- `hostname`: The device hostname
- `id`: The subscription identifier
- `initialized`: Whether the subscription has been initialized
- `ip`: The BMC IP address
- `kind`: The device type (`GB300` for compute trays, `GB200SW` for NVLink switches)
- `port`: The Redfish API port (typically 443 for HTTPS)
- `ssl`: Whether SSL/TLS is enabled
- `subscribed`: Whether the subscription is active
- `uuid`: The unique identifier for the device

N.2.5 Redfish Event Testing: `redfishevent`

The `redfishevent` command can be used to test Redfish event delivery. The `-m|--message` option sends a custom message that can be used to verify that event subscriptions are working correctly:

Example

```
[basecm11->device]% redfishevent -m test1
Tue Jan 13 05:40:27 2026 [warning] a07-p01-dgx-02-c05: test1
For details type: events details 7
```

Running the command multiple times triggers test events from different subscribed devices, confirming that Redfish event delivery is functioning across the cluster. The events are logged and can be viewed using the `events` command.

The `redfishevent` command has options for targeting specific nodes. Details can be seen by running the `help` command (`help redfishevent`).

N.2.6 Redfish HTTP Commands

BCM provides commands for direct Redfish API access using standard HTTP methods. These commands allow administrators to query and modify device configurations directly through the Redfish interface.

Redfish GET: `redfishget`

The `redfishget` command carries out an HTTP GET request to a Redfish endpoint. This is used to retrieve information from devices:

Example

```
[basecm11->device]% redfishget -r a07 /redfish/v1/Chassis
==== a07-p01-dgx-02-c02, curl code: 0, http code: 200 ====
{
  "@odata.id": "/redfish/v1/Chassis",
  "@odata.type": "#ChassisCollection.ChassisCollection",
  "Members": [
    { "@odata.id": "/redfish/v1/Chassis/BMC_0" },
```

```
    { "@odata.id": "/redfish/v1/Chassis/CBC_0" },
    { "@odata.id": "/redfish/v1/Chassis/Chassis_0" },
    ...
  ]
}
```

Redfish POST: `redfishpost`
The `redfishpost` command carries out an HTTP POST request to a Redfish endpoint. This is used to create resources or trigger actions. The `-f|--filename` option specifies a JSON template file, and `-p|--parameter` option is used for parameter substitutions:

Example

```
[basecm11->device]% redfishpost -r rack01 /redfish/v1/Chassis/Chassis_0/Sensors/LeakDetector

    -f /cm/local/examples/cmd/redfish/sensor-leak-threshold.json -p reading=0.5
```

Redfish PUT: `redfishput`
The `redfishput` command carries out an HTTP PUT request to a Redfish endpoint. This is used to replace an entire resource:

Example

```
[basecm11->device]% redfishput -n node001 /redfish/v1/Systems/System_0

    -f /cm/local/examples/cmd/redfish/system-config.json
```

Redfish PATCH: `redfishpatch`
The `redfishpatch` command carries out an HTTP PATCH request to a Redfish endpoint. This is used to partially update a resource:

Example

```
[basecm11->device]% redfishpatch -n node001 /redfish/v1/Systems/System_0

    -f /cm/local/examples/cmd/redfish/system-update.json
```

Redfish DELETE: `redfishdelete`
The `redfishdelete` command carries out an HTTP DELETE request to a Redfish endpoint. This is used to remove resources:

Example

```
[basecm11->device]% redfishdelete -n node001 /redfish/v1/EventService/Subscriptions/1
```

All Redfish HTTP commands support filtering by node, category, rack, and other criteria. Details can be seen by running the `help` command (for example: `help redfishget`).