

RIMA Cluster Manager 9.2

Administrator Manual

Revision: f502c1099

Date: Wed May 28 2025

©2024 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. PBS Professional, and Green Provisioning are trademarks 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 RIMA Cluster 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 RIMA Cluster 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 RIMA Cluster 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 RIMA Cluster 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 i
0.1 Quickstart xiii
0.2 About This Manual xiii
0.3 About The Manuals In General xiii
0.4 Getting Administrator-Level Support xiv
0.5 Getting Professional Services xiv

1 Introduction **1**
1.1 RIMA Cluster Manager Functions And Aims 1
1.2 The Scope Of The Administrator Manual (This Manual) 1
1.2.1 Installation 1
1.2.2 Configuration, Management, And Monitoring Via The Cluster Manager Tools And Applications 2
1.3 Outside The Direct Scope Of The Administrator Manual 2

2 Cluster Management With RIMA Cluster Manager **5**
2.1 Concepts 5
2.1.1 Devices 5
2.1.2 Software Images 6
2.1.3 Node Categories 7
2.1.4 Node Groups 8
2.1.5 Roles 8
2.1.6 Configuration Overlay 8
2.2 Modules Environment 9
2.2.1 Adding And Removing Modules 9
2.2.2 Using Local And Shared Modules 10
2.2.3 Setting Up A Default Environment For All Users 10
2.2.4 Creating A Modules Environment Module 11
2.2.5 Lua Modules Environment (LMod) 11
2.3 Authentication 12
2.3.1 Changing Administrative Passwords On The Cluster 12
2.3.2 Logins Using `ssh` 13
2.3.3 Certificates 14
2.3.4 Profiles 15
2.4 RIMA View GUI 15
2.4.1 Installing The Cluster Management GUI Service 15
2.4.2 Navigating The Cluster With RIMA View 17
2.5 Cluster Management Shell 20
2.5.1 Invoking `cmsh` 20
2.5.2 Levels, Modes, Help, And Commands Syntax In `cmsh` 24
2.5.3 Working With Objects 28

2.5.4 Accessing Cluster Settings . . . 39
2.5.5 Advanced cmsh Features . . . 39
2.6 Cluster Management Daemon . . . 52
2.6.1 Controlling The Cluster Management Daemon . . . 52
2.6.2 Configuring The Cluster Management Daemon . . . 53
2.6.3 CMDaemon Versions . . . 53
2.6.4 Configuring The Cluster Management Daemon Logging Facilities . . . 54
2.6.5 Configuration File Modification, And The FrozenFile Directive . . . 55
2.6.6 Configuration File Conflicts Between The Standard Distribution And The Cluster Manager For Generated And Non-Generated Files . . . 56
2.6.7 CMDaemon Lite . . . 56

3 Configuring The Cluster 59
3.1 Main Cluster Configuration Settings . . . 59
3.1.1 Cluster Configuration: Various Name-Related Settings . . . 60
3.1.2 Cluster Configuration: Some Network-Related Settings . . . 61
3.1.3 Miscellaneous Settings . . . 63
3.1.4 Limiting The Maximum Number Of Open Files . . . 66
3.2 Network Settings . . . 67
3.2.1 Configuring Networks . . . 68
3.2.2 Adding Networks . . . 71
3.2.3 Changing Network Parameters . . . 72
3.2.4 Tools For Viewing Cluster Connections . . . 83
3.3 Configuring Bridge Interfaces . . . 86
3.4 Configuring VLAN interfaces . . . 87
3.4.1 Configuring A VLAN Interface Using cmsh . . . 87
3.4.2 Configuring A VLAN Interface Using RIMA View . . . 88
3.5 Configuring Bonded Interfaces . . . 88
3.5.1 Adding A Bonded Interface . . . 88
3.5.2 Single Bonded Interface On A Regular Node . . . 89
3.5.3 Multiple Bonded Interface On A Regular Node . . . 90
3.5.4 Bonded Interfaces On Head Nodes And HA Head Nodes . . . 90
3.5.5 Tagged VLAN On Top Of a Bonded Interface . . . 90
3.5.6 Association Of MAC Address With A Bonded Interface . . . 91
3.5.7 Further Notes On Bonding . . . 91
3.6 Configuring InfiniBand And Omni-Path Interfaces . . . 92
3.6.1 Installing Software Packages . . . 93
3.6.2 Subnet Managers . . . 93
3.6.3 InfiniBand Network Settings . . . 94
3.6.4 Enabling SNMP . . . 95
3.6.5 Verifying Connectivity . . . 95
3.7 Configuring BMC (IPMI/iLO/DRAC/CIMC/Redfish) Interfaces . . . 97
3.7.1 BMC Network Settings . . . 97
3.7.2 BMC Authentication . . . 99
3.7.3 Interfaces Settings . . . 100
3.7.4 Identification With A BMC . . . 101
3.8 Configuring Switches And PDUs . . . 101

3.8.1 Configuring With The Manufacturer's Configuration Interface . . . 101
3.8.2 Configuring SNMP . . . 102
3.8.3 Uplink Ports . . . 105
3.8.4 The `showport` MAC Address to Port Matching Tool . . . 106
3.8.5 Disabling Port Detection . . . 106
3.8.6 The `switchoverview` Command . . . 107
3.9 Disk Layouts: Disked, Semi-Diskless, And Diskless Node Configuration . . . 108
3.9.1 Disk Layouts . . . 108
3.9.2 Disk Layout Assertions . . . 108
3.9.3 Changing Disk Layouts . . . 108
3.9.4 Changing A Disk Layout From Disked To Diskless . . . 108
3.10 Configuring NFS Volume Exports And Mounts . . . 110
3.10.1 Exporting A Filesystem Using RIMA View And `cmsh` . . . 112
3.10.2 Mounting A Filesystem Using RIMA View And `cmsh` . . . 115
3.10.3 Mounting A Filesystem Subtree For A Diskless Node Over NFS . . . 118
3.10.4 Mounting The Root Filesystem For A Diskless Node Over NFS . . . 120
3.10.5 Configuring NFS Volume Exports And Mounts Over RDMA With OFED Drivers . 122
3.11 Managing And Configuring Services . . . 123
3.11.1 Why Use The Cluster Manager For Services? . . . 123
3.11.2 Managing And Configuring Services—Examples . . . 124
3.12 Managing And Configuring A Rack . . . 129
3.12.1 Racks . . . 129
3.12.2 Assigning Devices To A Rack . . . 131
3.12.3 Assigning Devices To A Chassis . . . 132
3.13 Configuring GPU Settings . . . 135
3.13.1 GPUs And GPU Units . . . 135
3.13.2 Configuring GPU Settings . . . 135
3.13.3 MIG Configuration . . . 143
3.14 Configuring Custom Scripts . . . 152
3.14.1 `custompowerscript` . . . 152
3.14.2 `custompingscript` . . . 152
3.14.3 `customremoteconsolescript` . . . 153
3.14.4 `sysinfo` Custom Scripts . . . 153
3.15 Cluster Configuration Without Execution By CMDaemon . . . 156
3.15.1 Cluster Configuration: The Bigger Picture . . . 156
3.15.2 Making Nodes Function Differently By Image . . . 157
3.15.3 Making All Nodes Function Differently From Normal Cluster Behavior With `FrozenFile` . . . 160
3.15.4 Adding Functionality To Nodes Via An `initialize` Or `finalize` Script . . . 160
3.15.5 Examples Of Configuring Nodes With Or Without CMDaemon . . . 161
3.16 Saving A Backup Of Configuration Files With `versionconfigfiles` . . . 162

4 Power Management **163**
4.1 Configuring Power Parameters . . . 163
4.1.1 PDU-based Power Control . . . 164
4.1.2 IPMI-Based Power Control . . . 167
4.1.3 Combining PDU- and IPMI-Based Power Control . . . 167

4.1.4 Custom Power Control . . . 167
4.1.5 Hewlett Packard iLO-Based Power Control . . . 169
4.1.6 Dell drac-based Power Control . . . 169
4.1.7 Redfish-Based and CIMC-Based Power Control . . . 169
4.2 Power Operations . . . 169
4.2.1 Power Operations Overview . . . 169
4.2.2 Power Operations With RIMA View . . . 170
4.2.3 Power Operations Through cmsh . . . 171
4.3 Monitoring Power . . . 175
4.4 Switch Configuration To Survive Power Downs . . . 175

5 Node Provisioning 177
5.1 Before The Kernel Loads . . . 177
5.1.1 PXE Booting . . . 177
5.1.2 iPXE Booting From A Disk Drive . . . 181
5.1.3 iPXE Booting Using InfiniBand . . . 181
5.1.4 Using PXE To Boot From The Drive . . . 182
5.1.5 Network Booting Without PXE On The ARMv8 Architecture . . . 182
5.1.6 Network Booting Protocol . . . 182
5.1.7 The Boot Role . . . 182
5.2 Provisioning Nodes . . . 183
5.2.1 Provisioning Nodes: Configuration Settings . . . 183
5.2.2 Provisioning Nodes: Role Setup With cmsh . . . 184
5.2.3 Provisioning Nodes: Role Setup With RIMA View . . . 185
5.2.4 Provisioning Nodes: Housekeeping . . . 187
5.3 The Kernel Image, Ramdisk And Kernel Modules . . . 192
5.3.1 Booting To A "Good State" Software Image . . . 192
5.3.2 Selecting Kernel Driver Modules To Load Onto Nodes . . . 192
5.3.3 InfiniBand Provisioning . . . 194
5.3.4 Omni-Path Provisioning . . . 196
5.3.5 VLAN Provisioning . . . 197
5.4 Node-Installer . . . 198
5.4.1 Requesting A Node Certificate . . . 199
5.4.2 Deciding Or Selecting Node Configuration . . . 201
5.4.3 Starting Up All Network Interfaces . . . 211
5.4.4 Determining Install-mode Type And Execution Mode . . . 213
5.4.5 Running Initialize Scripts . . . 218
5.4.6 Checking Partitions, RAID Configuration, Mounting Filesystems . . . 218
5.4.7 Synchronizing The Local Drive With The Software Image . . . 219
5.4.8 Writing Network Configuration Files . . . 223
5.4.9 Creating A Local /etc/fstab File . . . 224
5.4.10 Booting From The Local Hard Drive . . . 224
5.4.11 Running Finalize Scripts . . . 226
5.4.12 Unloading Specific Drivers . . . 227
5.4.13 Switching To The Local init Process . . . 227
5.5 Node States . . . 227
5.5.1 Node States Icons In RIMA View . . . 227

5.5.2 Node States Shown In cmsh . . . 228
5.5.3 Node States Indicating Regular Start Up . . . 228
5.5.4 Node States That May Indicate Problems . . . 229
5.6 Updating Running Nodes . . . 231
5.6.1 Updating Running Nodes: Configuration With excludelistupdate . . . 232
5.6.2 Updating Running Nodes: With cmsh Using imageupdate . . . 239
5.6.3 Updating Running Nodes: With RIMA View Using the Update node Option . . . 239
5.6.4 Updating Running Nodes: Considerations . . . 239
5.7 Adding New Nodes . . . 240
5.7.1 Adding New Nodes With cmsh And RIMA View Add Functions . . . 240
5.7.2 Adding New Nodes With The Node Creation Wizard . . . 240
5.8 Troubleshooting The Node Boot Process . . . 241
5.8.1 Node Fails To PXE Boot . . . 242
5.8.2 Node-installer Logging . . . 246
5.8.3 Provisioning Logging . . . 247
5.8.4 Ramdisk Fails During Loading Or Sometime Later . . . 247
5.8.5 Ramdisk Cannot Start Network . . . 247
5.8.6 Node-Installer Cannot Create Disk Layout . . . 248
5.8.7 Node-Installer Cannot Start BMC (IPMI/iLO) Interface . . . 251

6 User Management 257
6.1 Managing Users And Groups With RIMA View . . . 257
6.2 Managing Users And Groups With cmsh . . . 259
6.2.1 Adding A User . . . 259
6.2.2 Saving The Modified State . . . 260
6.2.3 Editing Properties Of Users And Groups . . . 261
6.2.4 Reverting To The Unmodified State . . . 264
6.2.5 Removing A User . . . 264
6.3 Using An External LDAP Server . . . 264
6.3.1 External LDAP Server Replication . . . 266
6.3.2 High Availability . . . 269
6.4 Tokens And Profiles . . . 269
6.4.1 Modifying Profiles . . . 271
6.4.2 Creation Of Custom Certificates With Profiles, For Users Managed By The Cluster Manager's Internal LDAP . . . 271
6.4.3 Creation Of Custom Certificates With Profiles, For Users Managed By An External LDAP . . . 275
6.4.4 Logging The Actions Of CMDaemon Users . . . 276
6.4.5 Creation Of Certificates For Nodes With cm-component-certificate . . . 276

7 Post-Installation Software Management 279
7.1 RIMA Cluster Manager Packages And Their Naming Convention . . . 280
7.1.1 The packages Command . . . 281
7.2 Managing Packages On The Head Node . . . 282
7.2.1 Managing RPM Or .deb Packages On The Head Node . . . 282
7.2.2 Installation Of Packages On The Head Node That Are Not .deb And Not .rpm Packages . . . 285

7.3 Kernel Management On A Head Node Or Image . . . 285
7.3.1 Installing A Standard Distribution Kernel Into An Image Or On A Head Node . . 285
7.3.2 Excluding Kernels And Other Packages From Updates . . . 286
7.3.3 Updating A Kernel In A Software Image . . . 287
7.3.4 Setting Kernel Options For Software Images . . . 288
7.3.5 Kernel Driver Modules . . . 289
7.4 Managing A Package In A Software Image And Running It On Nodes . . . 291
7.4.1 Installing From Head Into The Image: Changing The Root Directory Into Which The Packages Are Deployed . . . 291
7.4.2 Installing From Head Into The Image: Updating The Node . . . 293
7.4.3 Installing From Head Into The Image: Possible Issues When Using `rpm --root`, `yum --installroot` Or `chroot` . . . 293
7.4.4 Managing A Package In The Node-Installer Image . . . 294
7.5 Managing Non-RPM Software In A Software Image And Running It On Nodes . . . 295
7.5.1 Managing The Software Directly On An Image . . . 295
7.5.2 Managing The Software Directly On A Node, Then Syncing Node-To-Image . . . 296
7.6 Creating A Custom Software Image . . . 299
7.6.1 Creating A Base Distribution Archive From A Base Host . . . 299
7.6.2 Creating The Software Image With `cm-create-image` . . . 301
7.6.3 Configuring Local Repositories For Linux Distributions, And For The Cluster Manager Package Repository, For A Software Image . . . 304
7.6.4 Creating A Custom Image From The Local Repository . . . 306
7.7 Creating Images For Other Distributions And Architectures (Multidistro And Multiarch) 307
7.7.1 The `cm-image` Tool . . . 307
7.7.2 Multidistro Examples: Provisioning From CentOS 7 Head Node To Ubuntu 18.04 Regular Nodes . . . 310
7.7.3 Multiarch Example: Creating An Image From A Centos 8 Head Node For ARMv8 Architecture Regular Nodes . . . 311

8 Monitoring: Monitoring Cluster Devices 315
8.1 A Basic Monitoring Example And Action . . . 315
8.1.1 Synopsis Of Basic Monitoring Example . . . 315
8.1.2 Before Using The Basic Monitoring Example—Setting Up The Pieces . . . 316
8.1.3 Using The Basic Monitoring Example . . . 317
8.2 Monitoring Concepts And Definitions . . . 320
8.2.1 Measurables . . . 320
8.2.2 Enummetrics . . . 323
8.2.3 Metrics . . . 324
8.2.4 Health Check . . . 325
8.2.5 Trigger . . . 327
8.2.6 Action . . . 327
8.2.7 Severity . . . 328
8.2.8 AlertLevel . . . 328
8.2.9 Flapping . . . 328
8.2.10 Data Producer . . . 328
8.2.11 Conceptual Overview: The Main Monitoring Interfaces Of RIMA View . . . 332
8.3 Monitoring Visualization With RIMA View . . . 333

8.3.1 The Monitoring Window . . . 333
8.4 Monitoring Configuration With RIMA View . . . 335
8.4.1 Monitoring Configuration: Data Producers . . . 336
8.4.2 Monitoring Configuration: Measurables . . . 338
8.4.3 Monitoring Configuration: Consolidators . . . 340
8.4.4 Monitoring Configuration: Actions . . . 344
8.4.5 Monitoring Configuration: Triggers . . . 347
8.4.6 Monitoring Configuration: Health status . . . 350
8.4.7 Monitoring Configuration: All Health Checks . . . 351
8.4.8 Monitoring Configuration: Standalone Monitored Entities . . . 351
8.4.9 Monitoring Configuration: PromQL Queries . . . 352
8.4.10 Monitoring Configuration: Resources . . . 352
8.4.11 Monitoring Configuration: Types . . . 353
8.5 The `monitoring` Mode Of `cmsh` . . . 354
8.5.1 The `action` Submode . . . 355
8.5.2 The `consolidator` Submode . . . 357
8.5.3 The `measurable` Submode . . . 359
8.5.4 The `setup` Submode . . . 363
8.5.5 The `standalone` Submode . . . 369
8.5.6 The `trigger` Submode . . . 369
8.6 Obtaining Monitoring Data Values . . . 372
8.6.1 Getting The List Of Measurables For An Entity: The `measurables`, `metrics`, `healthchecks` And `enummetrics` Commands . . . 372
8.6.2 On-Demand Metric Sampling And Health Checks . . . 373
8.6.3 The Latest Data And Counter Values—The `latest*data` And `latestmetriccounters` Commands . . . 376
8.6.4 Data Values Over A Period—The `dumpmonitoringdata` Command . . . 378
8.6.5 Monitoring Data Health Overview–The `healthoverview` Command . . . 386
8.6.6 Monitoring Data About The Monitoring System—The `monitoringinfo` Command 386
8.6.7 Dropping Monitoring Data With The `monitoringdrop` Command . . . 387
8.6.8 Monitoring Suspension And Resumption—The `monitoringsuspend` And `monitoringresume` Commands . . . 389
8.6.9 Monitoring Pickup Intervals . . . 390
8.7 Offloaded Monitoring . . . 392
8.7.1 Why Offloaded Monitoring? . . . 392
8.7.2 Implementing Offloaded Monitoring . . . 393
8.7.3 Background Details . . . 393
8.8 The User Portal . . . 397
8.8.1 Accessing The User Portal . . . 397
8.8.2 Setting A Common Username/Password For The User Portal . . . 398
8.8.3 User Portal Access . . . 399
8.8.4 User Portal Home Page . . . 399
8.9 Cloud Job Tagging . . . 400
8.10 Event Viewer . . . 401
8.10.1 Viewing Events In RIMA View . . . 401
8.10.2 Viewing Events In `cmsh` . . . 401

8.10.3 Using The Event Bucket From The Shell For Events And For Tagging Device States 403
8.10.4 InfoMessages . . . 403
8.11 Monitoring Location With GNSS . . . 405
8.12 Monitoring Report Queries . . . 406
8.12.1 Monitoring Report Queries In cmsh . . . 406
8.12.2 Monitoring Report Queries In RIMA View . . . 407

9 Day-to-day Administration 409
9.1 Parallel Shells: pdsh And pexec . . . 409
9.1.1 pdsh In The OS Shell . . . 410
9.1.2 pexec In cmsh . . . 413
9.1.3 pexec In RIMA View . . . 414
9.1.4 Using The -j|--join Option Of pexec In cmsh . . . 414
9.1.5 Other Parallel Commands . . . 415
9.2 Getting Support With Cluster Manager Issues, And Notifications For Release Updates . . . 415
9.2.1 Support Via E-mail . . . 416
9.2.2 Reporting Cluster Manager Diagnostics With cm-diagnose . . . 417
9.2.3 Requesting Remote Support With request-remote-assistance . . . 418
9.2.4 Getting Notified About Updates . . . 419
9.3 Backups . . . 420
9.3.1 Cluster Installation Backup . . . 420
9.3.2 Local Database And Data Backups And Restoration . . . 421
9.4 Revision Control For Images . . . 422
9.4.1 Btrfs: The Concept And Why It Works Well In Revision Control For Images . . . 422
9.4.2 Btrfs Availability And Distribution Support . . . 423
9.4.3 Installing Btrfs To Work With Revision Control Of Images In The Cluster Manager 423
9.4.4 Using cmsh For Revision Control Of Images . . . 425
9.5 BIOS Configuration And Updates . . . 428
9.5.1 BIOS Configuration Via CMDaemon And Redfish . . . 428
9.5.2 Updating BIOS Versions . . . 434
9.6 Hardware Match Check With The hardware-profile Data Producer . . . 436
9.7 Serial Over LAN Console Access . . . 438
9.7.1 Background Notes On Serial Console And SOL . . . 438
9.7.2 SOL Console Configuration With RIMA View . . . 440
9.7.3 SOL Console Configuration And Access With cmsh . . . 440
9.7.4 The conman Serial Console Logger And Viewer . . . 441
9.8 Managing Raw Monitoring Data . . . 445
9.8.1 Monitoring Subsystem Disk Usage With The monitoringinfo --storage Option 445
9.8.2 Estimating The Required Size Of The Storage Device . . . 445
9.8.3 Moving Monitoring Data Elsewhere . . . 446
9.8.4 Reducing Monitoring Data By Reducing Samples . . . 446
9.8.5 Deleting All Monitoring Data . . . 446
9.9 Node Replacement . . . 448
9.10 Ansible And RIMA Cluster Manager . . . 448
9.10.1 An Overview Of Ansible . . . 448
9.10.2 A Simple Playbook Example . . . 449

9.10.3 An Intermediate Playbook Example: Setting Up A Cluster For Demonstration Purposes . . . 451
9.10.4 A More Complicated Playbook Example: Creating An Edge Site And Related Properties . . . 452

10 High Availability **461**
10.0 Introduction . . . 461
10.0.1 Why Have High Availability? . . . 461
10.0.2 High Availability—For What Nodes? . . . 461
10.0.3 High Availability Usually Uses Shared Storage . . . 462
10.0.4 Organization Of This Chapter . . . 462
10.1 HA Concepts . . . 462
10.1.1 Primary, Secondary, Active, Passive . . . 462
10.1.2 Monitoring The Active Head Node, Initiating Failover . . . 462
10.1.3 Services In The Cluster Manager HA Setups . . . 463
10.1.4 Failover Network Topology . . . 464
10.1.5 Shared Storage . . . 465
10.1.6 Guaranteeing One Active Head At All Times . . . 466
10.1.7 Automatic Vs Manual Failover . . . 467
10.1.8 HA And Cloud Nodes . . . 468
10.1.9 HA Using Virtual Head Nodes . . . 468
10.2 HA Setup Procedure Using `cmha-setup` . . . 468
10.2.1 Preparation . . . 469
10.2.2 Failover Cloning (Replacing A Passive Head) . . . 471
10.2.3 Shared Storage Setup . . . 474
10.2.4 Automated Failover And Relevant Testing . . . 476
10.3 Running `cmha-setup` Without ncurses, Using An XML Specification . . . 477
10.3.1 Why Run It Without ncurses? . . . 477
10.3.2 The Syntax Of `cmha-setup` Without ncurses . . . 477
10.3.3 Example `cmha-setup` Run Without ncurses . . . 478
10.4 Managing HA . . . 478
10.4.1 Changing An Existing Failover Configuration . . . 479
10.4.2 `cmha` Utility . . . 479
10.4.3 States . . . 482
10.4.4 Failover Action Decisions . . . 483
10.4.5 Keeping Head Nodes In Sync . . . 484
10.4.6 High Availability Parameters . . . 486
10.4.7 Viewing Failover Via RIMA View . . . 488
10.4.8 Re-cloning A Head Node . . . 488
10.5 HA For Regular Nodes And Edge Director Nodes . . . 489
10.5.1 Why Have HA On Non-Head Nodes? . . . 489
10.5.2 Comparing HA For Head Nodes, Regular Nodes And Edge Director Nodes . . . 489
10.5.3 Setting Up A Regular Node HA Service . . . 490
10.5.4 The Sequence Of Events When Making Another HA Regular Node Active . . . 494
10.6 HA And Workload Manager Jobs . . . 495

A Generated Files **497**
A.1 System Configuration Files Created Or Modified By CMDeamon On Head Nodes 497
A.2 System Configuration Files Created Or Modified By CMDaemon In Software Images: . . 500
A.3 System Configuration Files Created Or Modified By CMDaemon Directly On The Node . 501
A.3.1 Options To `filewriteinfo` 501
A.4 Files Created On Regular Nodes By The Node-Installer 502
A.5 Files Not Generated, But Installed In RHEL And Derivatives 503

B RIMA Public Key **509**

C CMDaemon Configuration File Directives **511**

D Disk Partitioning And RAID Configuration **541**
D.1 Structure Of Partitioning Definition—The Global Partitioning XML Schema File 541
D.2 Structure Of Hardware RAID Definition—The Hardware RAID XML Schema File 548
D.3 Example: Default Node Partitioning 551
D.4 Example: Hardware RAID Configuration 553
D.4.1 RAID level 0 And RAID 10 Example 553
D.5 Example: Software RAID 555
D.6 Example: Software RAID With Swap 556
D.7 Example: Logical Volume Manager 557
D.8 Example: Logical Volume Manager With RAID 1 558
D.9 Example: Diskless 560
D.10 Example: Semi-diskless 560
D.11 Example: Preventing Accidental Data Loss 561
D.12 Example: Using Custom Assertions 562
D.13 Example: Software RAID1 With One Big Partition 563
D.14 Example: Software RAID5 With One Big Partition 565
D.15 Example: Software RAID1 With Standard Partitioning 567
D.16 Example: Software RAID5 With Standard Partitioning 569
D.17 Example: LUKS Disk Encryption With Standard Partitioning 572
D.17.1 Introduction 572
D.17.2 Node Provisioned Over The Network: Encrypted Partition XML Example 573
D.17.3 Standalone Node: Encrypted Partition XML Example 576
D.17.4 Changing A Passphrase On An Encrypted Node 576

E Example `initialize` And `finalize` Scripts **579**
E.1 When Are They Used? 579
E.2 Accessing From RIMA View And `cmsh` 579
E.3 Environment Variables Available To `initialize` And `finalize` Scripts 580
E.4 Using Environment Variables Stored In Multiple Variables 583
E.5 Storing A Configuration To A Filesystem 584
E.5.1 Storing With Initialize Scripts 584
E.5.2 Ways Of Writing A Finalize Script To Configure The Destination Nodes 584
E.5.3 Restricting The Script To Nodes Or Node Categories 587

F Metrics, Health Checks, Enummetrics, And Actions **589**
F.1 Metrics And Their Parameters . . . 589
F.1.1 Regular Metrics . . . 590
F.1.2 NFS Metrics . . . 595
F.1.3 InfiniBand Metrics . . . 597
F.1.4 Monitoring System Metrics . . . 598
F.1.5 GPU Metrics . . . 600
F.1.6 Job Metrics . . . 601
F.1.7 IPMI Metrics . . . 606
F.1.8 SMART Metrics . . . 607
F.1.9 Prometheus Metrics . . . 608
F.1.10 Parameters For Metrics . . . 611
F.2 Health Checks And Their Parameters . . . 614
F.2.1 Regular Health Checks . . . 614
F.2.2 Redfish Healthchecks . . . 619
F.2.3 Parameters For Health Checks . . . 619
F.3 Enummetrics . . . 620
F.4 Actions And Their Parameters . . . 621
F.4.1 Actions . . . 621
F.4.2 Parameters For A Monitoring Action . . . 621

G Changing The LDAP Password **623**
G.1 Setting A New Password For The LDAP Server . . . 623
G.2 Setting The New Password In `cmd.conf` . . . 623
G.3 Checking LDAP Access . . . 624

H Tokens **625**

I Understanding Consolidation **633**
I.1 Introduction . . . 633
I.2 What Is Consolidation? . . . 633
I.3 Raw Data And Consolidation . . . 633
I.4 A Demonstration Of The Output . . . 634

J Node Execution Filters And Execution Multiplexers **637**
J.1 Data Producers: Default Configuration For Running And Sampling . . . 638
J.1.1 Nodes That Data Producers Are Running On By Default—The `nodes` Command . . . 638
J.1.2 Nodes That Data Producers Target By Default—The `samplenow` Command . . . 638
J.2 Data Producers: Configuration For Running And Targeting . . . 639
J.2.1 Custom Metrics From The `fm.sh` Custom Script . . . 639
J.3 Replacing A Resource With An Explicit Node Specification . . . 641
J.4 Excessive Sampling . . . 642
J.5 Not Just For Nodes . . . 643
J.6 Lua Node Execution Filters . . . 643

K A Tree View Of `cmsh` **647**
K.1 Modes . . . 647

Preface

Welcome to the *Administrator Manual* for the RIMA Cluster Manager 9.2 cluster environment.

0.1 Quickstart

For readers who want to get a cluster up and running as quickly as possible with RIMA Cluster 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 the cluster manager so as to get the best out of it.

The *Administrator Manual* covers administration topics which are specific to the cluster manager 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 RIMA Cluster Manager 9.2 manuals are available on updated clusters by default at `/cm/shared/docs/cm`. The latest updates are always online at `http://support.rima.com/manuals`.

- 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 program with the cluster manager.
- The *Machine Learning Manual* describes how to install and configure machine learning capabilities with the cluster manager.
- The *Edge Manual* describes how to install and configure machine learning capabilities with the cluster manager.
- The *Containerization Manual* describes how to manage containers with the cluster manager.

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 cluster manager environment and the addition of new hardware and/or applications. The manuals also regularly incorporate customer feedback. Administrator and user input is greatly valued at RIMA. So any comments, suggestions or corrections will be very gratefully accepted at `manuals@rima.com`.

There is also a feedback form available via RIMA View, via the Menu icon,  , following the clickpath: →`Help`→`Feedback Form`

0.4 Getting Administrator-Level Support

If the reseller from whom the cluster manager was bought offers direct support, then the reseller should be contacted.

Otherwise the primary means of support is via the website `https://support.brightcomputing.com`. This allows the administrator to submit a support request via a web form, and opens up a trouble ticket. It is a good idea to try to use a clear subject header, since that is used as part of a reference tag as the ticket progresses. Also helpful is a good description of the issue. The followup communication for this ticket typically goes via standard e-mail. Section 9.2 has more details on working with support.

0.5 Getting Professional Services

RIMA normally differentiates between

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

Professional services can be provided after consulting with the reseller, or the RIMA account manager.

1

Introduction

1.1 RIMA Cluster Manager Functions And Aims

RIMA Cluster Manager contains tools and applications to facilitate the installation, administration, and monitoring of a cluster. In addition, the cluster manager 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 the cluster manager, 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 the cluster manager:** This is covered in the *Installation Manual*, which gives a short introduction to the concept of a cluster along with details on installing the cluster manager onto the head node. The *Installation Manual* is therefore the first manual an administrator should usually turn to when getting to work with the cluster manager for the first time. The *Administrator Manual* can be referred to as the main reference resource once the head node has had the cluster manager installed on it.

- **Provisioning installation:** This is covered in the *Administrator Manual*. After the head node has had the cluster manager 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 the cluster manager is described in detail in Chapter 7 of this manual.

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

- **Cloudbursting, Edge, and Machine Learning**: these are integrated by RIMA in various ways. These have their own deployment procedures and have separate manuals.

1.2.2 Configuration, Management, And Monitoring Via The Cluster Manager Tools And Applications

The administrator normally deals with the cluster software configuration via a front end to the cluster manager. This can be GUI-based (RIMA View, section 2.4) or shell-based (`cmsh`, section 2.5). Other tasks can be handled via special tools provided with the cluster manager, or the usual Linux tools. The use of the cluster manager 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 RIMA Cluster Manager	Introduction to main concepts and tools of the cluster manager. Lays down groundwork for the remaining chapters
3 Configuring The Cluster	Further configuration and set up of the cluster after software installation of the cluster manager 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 Post-Installation Software Management	Managing, updating, modifying the cluster manager software and images
8 Monitoring: Monitoring Cluster Devices	Device monitoring and conditional action triggers
9 Day-To-Day Administration	Miscellaneous administration
10 High Availability	Background details and setup instructions to build a cluster with redundant head nodes

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

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

- Cloudbursting (*Cloudbursting Manual*)
- Machine Learning (*Machine Learning Manual*)
- Edge deployment (*Edge Manual*)
- Developer topics (*Developer Manual*)
- Containerization topics (*Containerization 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 knowledge base** at `http://kb.brightcomputing.com` often supplements the *Administrator Manual* with discussion of the following:

  - Obscure, or complicated, configuration cases
  - Procedures that are not really within the scope of the cluster manager itself, but that may come up as part of related general Linux configuration.

- **Further support options.** If the issue is not described adequately in this manual, then section 9.2 describes how to get further support.

2

Cluster Management With RIMA Cluster Manager

This chapter introduces cluster management with RIMA Cluster Manager. A cluster running the cluster manager 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 the cluster manager.

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 (RIMA 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 the cluster manager are introduced.

2.1.1 Devices

A *device* in the cluster manager 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
- Ethernet Switch
- InfiniBand Switch
- Lite Node

- Myrinet Switch
- 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 the cluster manager, 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 the cluster manager has a device state associated with it. The table below describes the most important states for devices:

device statuses	device is	monitored by RIMA?	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 cluster manager 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 7, CentOS 7, and Scientific Linux 7 mixtures, are completely compatible with each other on the head and regular nodes. The same applies to RHEL8 and Rocky Linux 8 mixtures. That is because CentOS, Scientific Linux, and Rocky Linux are designed to be binary-compatible derivatives of their 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 7.

2.1.3 Node Categories

Reasons For Categories

The collection of settings in the cluster manager that can apply to a node is called the configuration of the node. The administrator usually configures nodes using the RIMA 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 77), 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 216) and `installbootrecord` (page 224), 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. This is reasonably 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 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 role, a category, or a configuration overlay.

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

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/9.2.0
module add openmpi/gcc/64/1.10.7
module add openblas
module add openblas/dynamic/0.2.20
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@rima92 ~]$ 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@rima92 ~]# 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@rima92 ~]# 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@rima92 ~]# 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 the cluster manager.

2.2.5 Lua Modules Environment (LMod)

By default, the cluster manager uses traditional Tcl scripts for its module files, or *TMod*. Lua modules, or *LMod*, provide an alternative modules environment, 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 RIMA 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@rima92 ~]# 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@rima92 ~]# 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 RIMA 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@rima92 ~]# 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 12).

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 G.

If the administrator has stored the password to the cluster in the RIMA 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 clickpath category→set usernodelogin, or the RIMA View clickpath Grouping→Categories[default]↓Edit→Settings→User node login
- 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.

The cluster manager 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 the cluster manager: 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 the cluster manager 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 275.

Using RIMA View and authenticating to the cluster manager: When an administrator uses the RIMA 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 clickpath is:
    about:preferences#privacy→Certificates→View Certificates→Your Certificates→Import

  The browser can then access the RIMA 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 6.4.

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 6.4.

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 6.4).

2.4 RIMA View GUI

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

RIMA View is supported to run on the latest 2 versions of Firefox, Google Chrome, Edge, and Safari. "Latest 2 versions" means the latest two publicly released versions at the time of release of RIMA Cluster Manager. For example, at the time of writing of this section, February 2023, the latest 2 versions were:

Browser	Versions
Chrome	108, 109
Edge	108, 109
Firefox	108, 109
Safari	16.2, 16.3

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

RIMA 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).

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

The cluster manager GUI interface—RIMA View—is provided by default as a web service on port 8081, from the head node to the browser. Its direct URL takes the form:

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

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

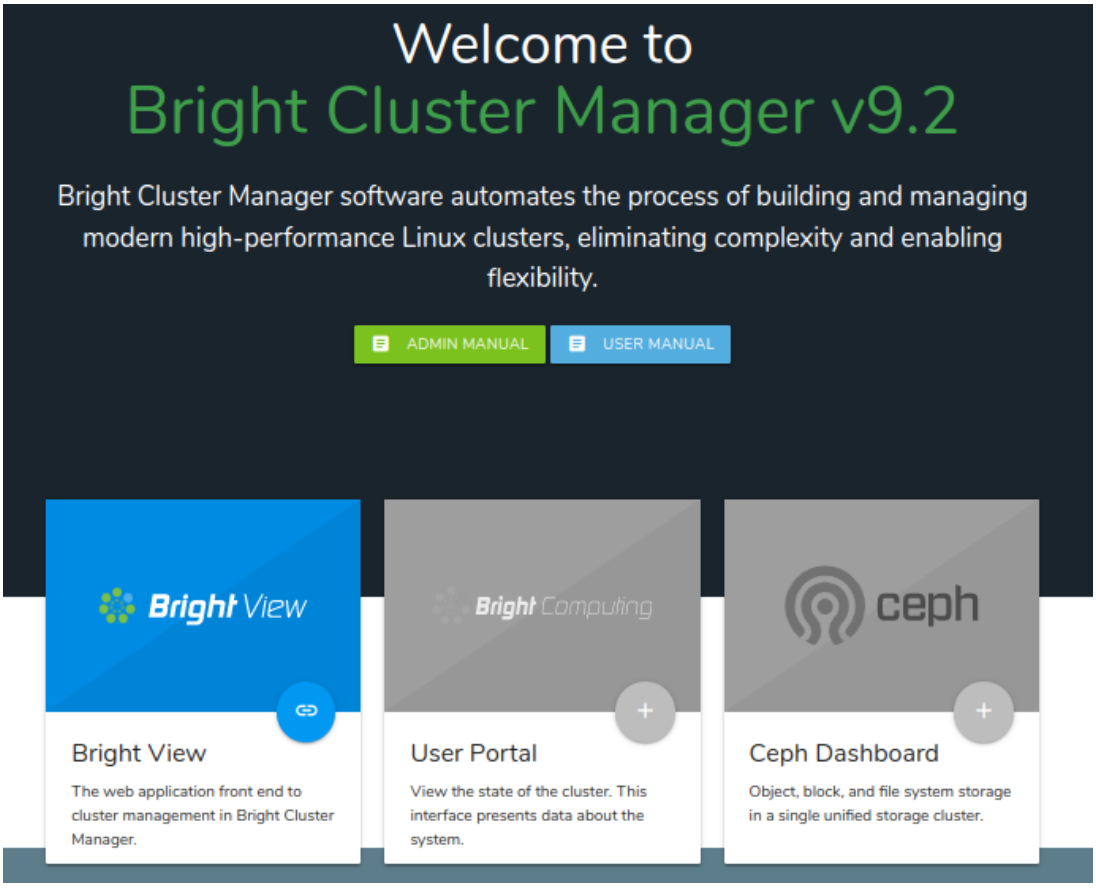


Figure 2.1: Head node hostname or IP address landing page at `https://`*<host name or IP address>*

RIMA Cluster Manager RIMA View Login Window

Figure 2.2 shows the login dialog window for RIMA View.

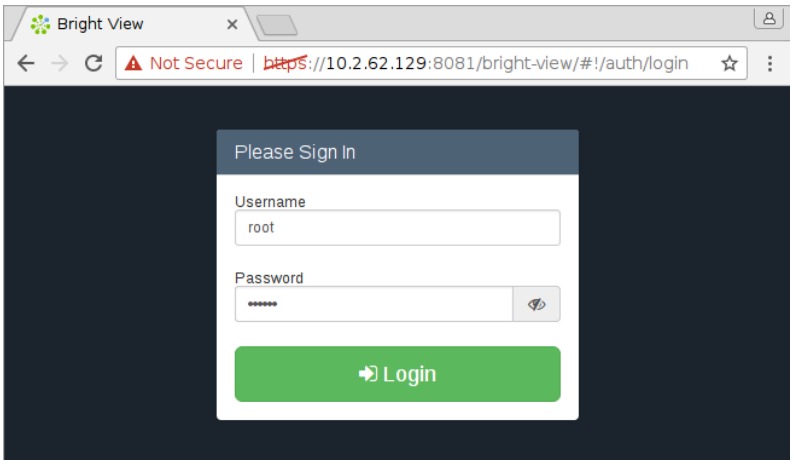


Figure 2.2: RIMA View Login via `https://`*<host name or IP address>*`:8081/rima-view`

RIMA Cluster Manager `RIMA View` Default Display On Connection

Clicking on the `Login` button logs the administrator into the RIMA View service on the cluster. By default an overview window is displayed, corresponding to the path `Cluster`→`Partition base` (figure 2.3).

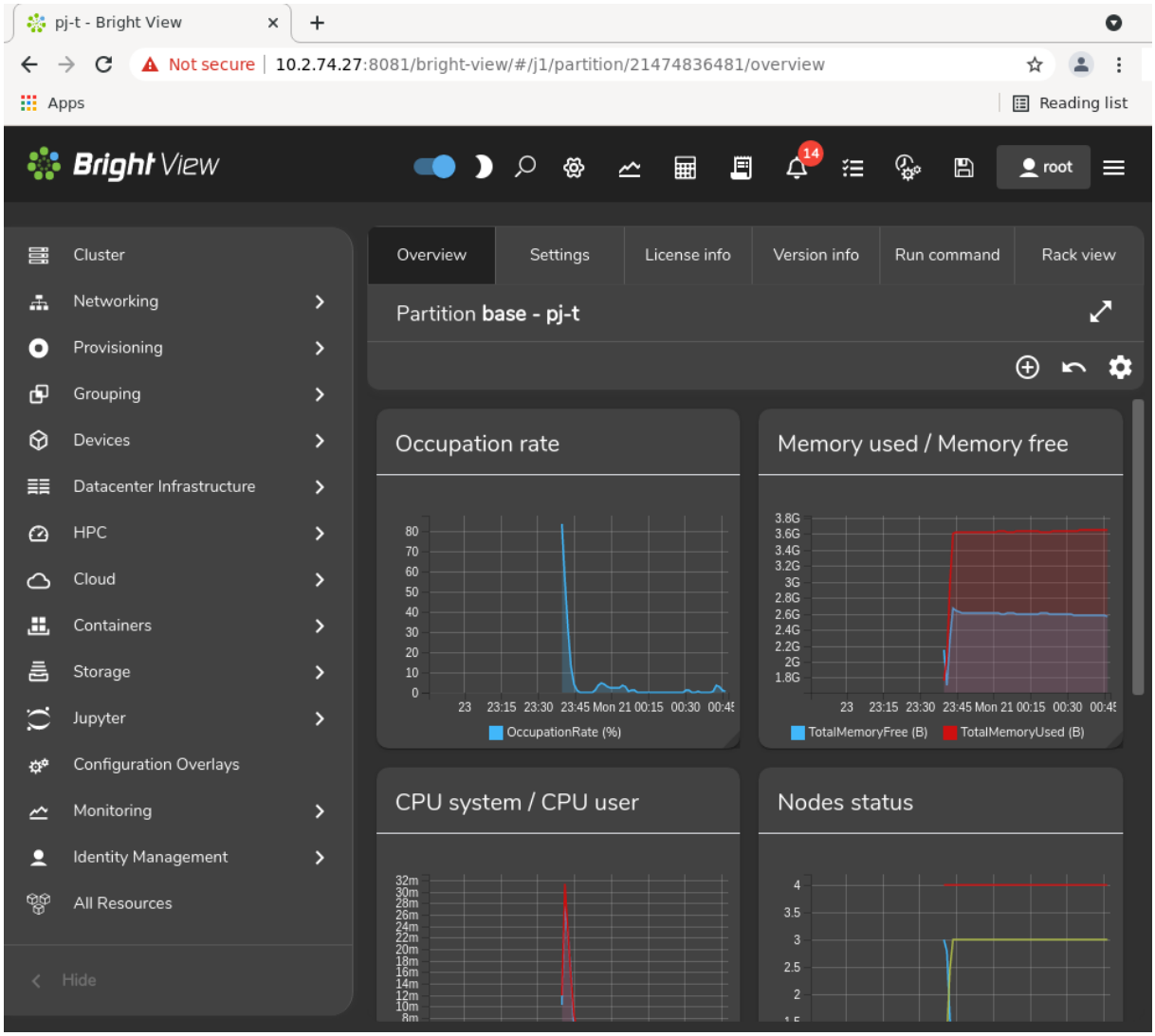


Figure 2.3: Cluster Overview

2.4.2 Navigating The Cluster With RIMA View

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

The resource tree, displayed on the left side of the window, consists of available cluster usage concepts such as `Provisioning`, `Grouping`, `HPC`, `Cloud`, and `Containers`. It also has a cluster-centric approach to miscellaneous system concepts such as hardware devices `Devices`, non-hardware resources such as `Identity Management`, and `Networking`.

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. This opens up the so-called `Partition base` window, which is essentially a representation of the cluster instance.[1]

The tabs within the `Partition base` window are mapped out in figure 2.4 and summarily described next.

[1]The name `Partition base` is literally a footnote in the cluster manager history. It derives from the time that RIMA clusters were planned to run in separate partitions within the cluster hardware. The primary cluster was then to be the `base` cluster, running in the base partition. The advent of the cluster manager cloud computing options in the form of the Cluster-On-Demand option (Chapter 2 of the *Cloudbursting Manual*), and the Cluster Extension option (Chapter 3 of the *Cloudbursting Manual*) means developing cluster partitions is no longer a priority.

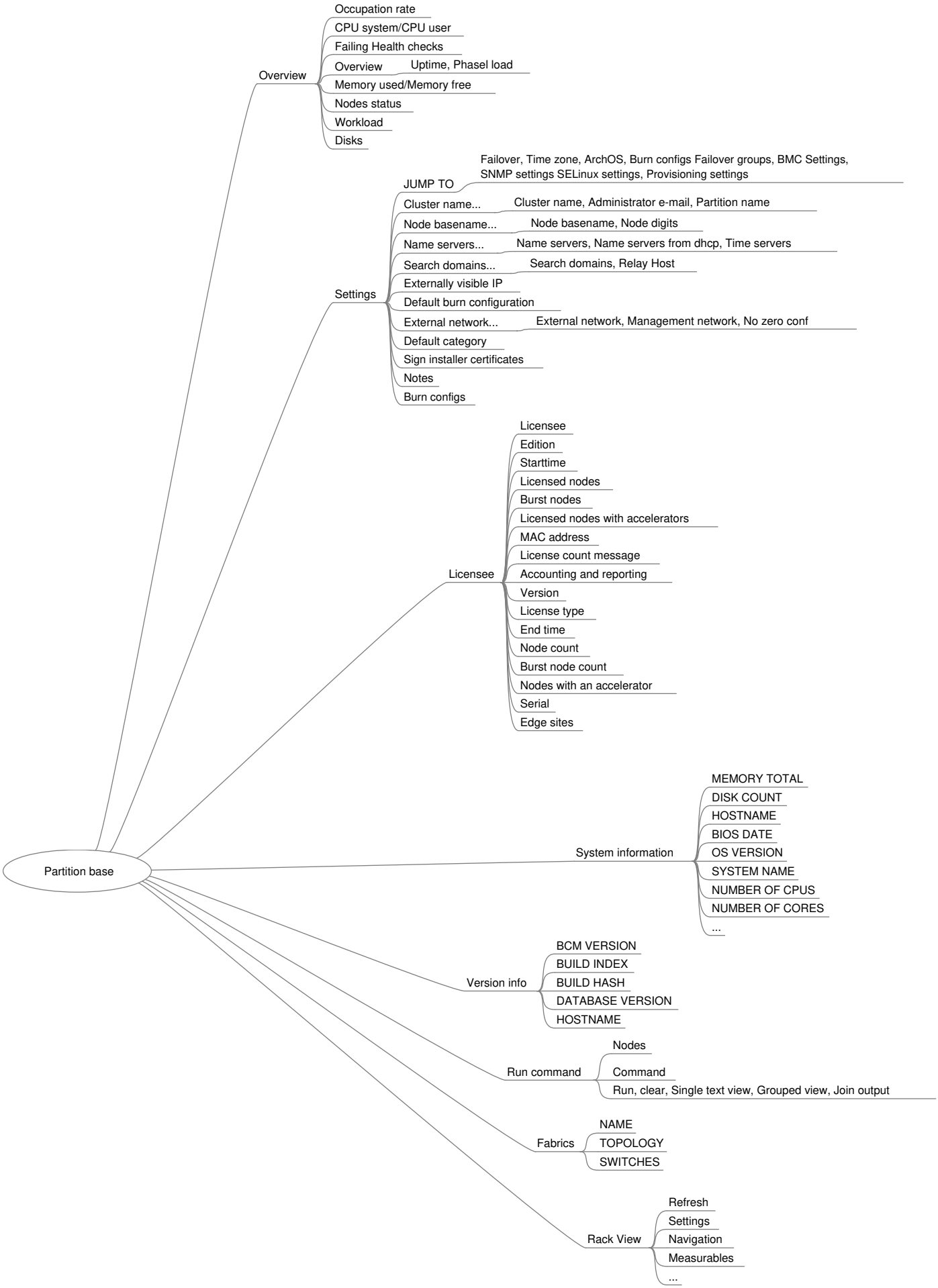


Figure 2.4: Cluster Navigation Within The Partition Base Window

Overview
The Overview tab (figure 2.3, page 17) shows the Occupation rate (page 592), memory used, CPU cycles used, node statuses, and other helpful cluster overview details.

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

- Buttons for jumping to settings for: Failover, time zone, ArchOS, burn configuration, failover groups, BMC settings, SNMP settings, SELinux settings, provisioning settings.
- Cluster name, Administrator e-mail, partition name
- Node basename, Node digits
- Name servers, Time servers
- Search domains, Relay Host
- Externally visible IP, Provisioning Node Auto Update Timeout
- Default burn configuration
- External network, Management network
- Default category: Sets the default category
- Sign installer certificates
- Notes

License info
The License info tab shows information to do with cluster licensing. A slightly obscure property within this window is Version, which refers to the version type of the license. The license for RIMA Cluster Manager version 7.0 and above is of a type that is compatible with versions all the way up to 9.2. RIMA Cluster Manager license versions from before 7.0 are not compatible. In practice it means that an upgrade from before 7.0, to 7.0 or beyond, requires a license upgrade. RIMA must be contacted to arrange the license upgrade.

System information
The System information tab shows the main hardware specifications of the node (CPU, memory, BIOS), along with the operating system version that it runs.

Version info
The Version info tab shows version information for important cluster software components, such as the CMDaemon database version, and the cluster manager version and builds.

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

Fabrics
The Fabrics tab displays the topology and switches for the fabrics used.

Rack View
The Rack View tab displays a view of the rack as defined by node allocations made by the administrator to racks and chassis.

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 the cluster manager. Since cmsh and RIMA 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 RIMA 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 cluster manager 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 dotfiles, if they exist, upon start-up in both batch and interactive mode. In the following list of dotfiles, a setting in the file that is in the shorter path will override a setting in the file with the longer path (i.e.: "shortest path overrides"):

- ~/.cm/cmsh/.cmshrc

- ~/.cm/.cmshrc
- ~/.cmshrc

If there is no dotfile for the user, then, if it exists, the file /etc/cmshrc, which is accessible to all users, is sourced, and its settings used.

If /etc/cmshrc exists, then its settings are used, but the values can be overridden by user dotfiles, which is standard Unix behavior.

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:

```
[rima92]% 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
```

Since RIMA Cluster Manager version 9.1, the cmsh session 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

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
```

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
beegfs ........................ Enter beegfs mode
category ...................... Enter category mode
ceph .......................... Enter ceph mode
cert .......................... Enter cert mode
cloud ......................... Enter cloud mode
cmjob ......................... Enter cmjob 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
keyvaluestore ................. Enter keyvaluestore mode
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
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
unmanagednodeconfiguration .... Enter unmanagednodeconfiguration mode
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
ceph .......................... Enter ceph 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 K.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 21).

  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

```
[rima92->configurationoverlay[slurm-client]->roles[slurmclient]]% list
Name (key)
----------------------------
slurmclient
[rima92->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:

```
[rima92 ~]# 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

```
[rima92]% device
[rima92->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         rima92        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 ]
[rima92->device]% exit
[rima92]% device
[rima92->device]% use node001
[rima92->device[node001]]% path
home;device;use node001;
[rima92->device[node001]]% home
[rima92]% home;device;use node001    #copy-pasted from path output earlier
[rima92->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. [2]

Example

```
[rima92->network]% device status rima92; list
rima92 ............ [   UP   ]
Name (key)    Type      Netmask bits  Base address  Domain name          Ipv6
------------- --------- ------------- ------------- -------------------- ----
externalnet   External  16            192.168.1.0   rima.com  no
globalnet     Global    0             0.0.0.0       cm.cluster
internalnet   Internal  16            10.141.0.0    eth.cluster
[rima92->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

```
[rima92->network]% device; status rima92; list
rima92 ................ [   UP   ]
Type          Hostname (key)  MAC                Category   Ip             Network     Status
------------- --------------- ------------------ ---------  -------------- ----------- ------
HeadNode      rima92        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 ]
[rima92->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 a new object
unassign	Unassign an object
clear	Clear the values of the object
clone	Clone the object and use it

...continues

[2] A more precise synopsis is:
[<mode>] <cmd> [<arg> ...] [; ... ; [<mode>] <cmd> [<arg> ...]]

...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. The IP address may also be set.

Usage:
 add <type> <hostname> [IP address]

Arguments:
 type
   chassis, genericdevice, gpuunit, litenode, cloudnode, physicalnode, headnode,
   powerdistributionunit, racksensor, ethernetswitch, ibswitch, myrinetswitch
```

`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 modified command corresponds roughly to the functionality of the Unsaved entities icon in figure 8.5.

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
[rima92->device[mycluster]->interfaces]% swap eth0 eth1; commit
[rima92->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, and corresponds roughly to the Revert button in RIMA View when viewing Unsaved entities (figure 8.5). 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

```
[rima92->device]% list -f hostname:14,switchports:15,ip
hostname (key) switchports     ip
-------------- --------------- --------------------
apc01                          10.142.254.1
rima92       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
[rima92->device]%
```

Running the `list` command with no argument uses the current format string for the mode.

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

```
[rima92->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

```
[rima92->device]% format hostname:10
[rima92->device]% list
hostname (
----------
apc01
rima92
node001
node002
switch01
```

A range of widths can be set, from a minimum to a maximum, using square brackets. A single minimum width possible is chosen from the range that fits all the characters of the column. If the number of characters in the column exceeds the maximum, then the maximum value is chosen. For example:

Example

```
[rima92->device]% format hostname:[10-14]
[rima92->device]% list
hostname (key)
--------------
apc01
rima92
node001
node002
switch01
```

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

```
[rima92->device]% format hostname:[10-14],switchports:14,ip:20
[rima92->device]% list
hostname (key) switchports    ip
-------------- -------------- --------------------
apc01                         10.142.254.1
rima92        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

```
[rima92->device]% format "hostname:[16-32], network:[14-32], status:[16-32]"
```

or

```
[rima92->device]% format hostname:[16-32],network:[14-32],status:[16-32]
```

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

Example

```
[rima92->device]% format -r
[rima92->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]
[rima92->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

```
[rima92->device]% sort type mac
[rima92->device]% list -f type:15,hostname:15,mac
type            hostname (key)  mac
--------------- --------------- --------------------
HeadNode        rima92        08:0A:27:BA:B9:43
PhysicalNode    node002         00:00:00:00:00:00
PhysicalNode    log001          52:54:00:DE:E3:6B
[rima92->device]% sort type hostname
[rima92->device]% list -f type:15,hostname:15,mac
type            hostname (key)  mac
--------------- --------------- --------------------
HeadNode        rima92        08:0A:27:BA:B9:43
PhysicalNode    log001          52:54:00:DE:E3:6B
PhysicalNode    node002         00:00:00:00:00:00

[rima92->device]% sort mac hostname
[rima92->device]% list -f type:15,hostname:15,mac
```

```
type            hostname (key)  mac
--------------- --------------- -------------------
PhysicalNode    node002           00:00:00:00:00:00
HeadNode        rima92          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:

```
[rima92->device]% list -f type:15,hostname:15,mac --sort type,mac
[rima92->device]% list -f type:15,hostname:15,mac --sort type,hostname
[rima92->device]% list -f type:15,hostname:15,mac --sort mac,hostname
```

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 229) 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
---------------------------------- ----------------------------------
Activation                         Thu, 03 Aug 2017 15:57:42 CEST
BMC Settings                       <submode>
Block devices cleared on next boot
Category                           default
...
Data node                          no
Default gateway                    10.141.255.254 (network: internalnet)
...
Software image                     default-image
Static routes                      <0 in submode>
...
```

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@rima92 ~]# cmsh
[rima92]% device roles node001
[rima92->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

```
[rima92->device[node001]->roles]% import<TAB><TAB>
cephmds     cloudgateway    elasticsearch
...and other available roles including  slurmclient...
[rima92->device[node001]->roles]% import --overlay slurm-client slurmclient
[rima92->device*[node001*]->roles*]% list
Name (key)
------------------------------
[category:default] cgroupsupervisor
slurmclient
[rima92->device*[node001*]->roles*]% set slurmclient queues node1q
[rima92->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 the cluster manager 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 the cluster manager, 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
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>
SELinux Settings               <submode>
Provisioning 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

```
[rima92->device[node001]]% history | tail -3
  162 use node001
  163 history
  164 history | tail -3
[rima92->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

```
[rima92]% device ssh node001
<screen is cleared>
<some MOTD text and login information is displayed>
[root@node001 ~]# exit
Connection to node001 closed.
[rima92]% device use rima92
[rima92->device[rima92]]% #now let us connect to the head node from the head node object
```

```
[rima92->device[rima92]]% ssh
<screen is cleared>
<some MOTD text and login information is displayed>
[root@rima92 ~]# exit
logout
Connection to rima92 closed.
[rima92->device[rima92]]%
```

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

```
[rima92->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

```
[rima92->device]% watch newnodes
 screen clears
Every 2.0s: newnodes                                    Thu Dec  3 13:01:45 2015
No new nodes currently available.
```

Example

```
[rima92->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 46) 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@rima92 ~]# 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|genericdevice|gpuunit|litenode|cloudnode|node|physicalnode|headnode|powerdistributionunit|racksensor|ethernetswitch|ibswitch|myrinetswitch|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 45) 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

```
[rima92->device]% foreach --clone node001 -n node022..node024 ()
[rima92->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
[rima92->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.

  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`, `ibswitch`.

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

  Example

```
[rima92->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

```
[rima92->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 ]
```

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

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.

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@rima92 ~]# 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

```
[rima92->device]% range -n node0[01-24]
[rima92->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

```
[rima92->device{-n node001..024}]% range -c default
[rima92->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

```
[rima92->device{-c default}]% get revision




[rima92->device{-c default}]% set revision test
[rima92->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 via a semi-colon (new command statement continuation on same line) syntax, as follows:

Example

```
[rima92->device{-c default}]% roles; assign pbsproclient; 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

```
[rima92->device{-c default}]% get ip
10.141.0.1
10.141.0.2
10.141.0.3
[rima92->device{-c default}]% ..
[rima92->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

```
[rima92->device]% range -n node[001-100]
[rima92->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@rima92 ~]# 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 20) 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@rima92 ~]# 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 RIMA View (section 3.1.1) or in `cmsh` (section 2.5.4, and also page 79).

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

```
[rima92->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

```
[rima92->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

```
[rima92->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

```
[rima92->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@rima92 ~]# cmsh
[rima92]% device
[rima92->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
[rima92->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@rima92 ~]# cmsh
[rima92]% device
[rima92->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
[rima92->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
[rima92->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`

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

```
[rima92->device]% list -f hostname,ip,mac,status
hostname (key)       ip                   mac                  status
-------------------- -------------------- -------------------- --------------------
node001              10.141.0.1           FA:16:3E:95:80:9F    [   UP   ]
rima92             10.141.255.254       FA:16:3E:D3:56:E0    [   UP   ]


[rima92->device]% color off; list -f hostname,ip,mac,status -d {}
[
  {
    "hostname (key)": "rima92",
    "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   ]"
  }
]
[rima92->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@rima92 ~]# 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 RIMA 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 9.8).

2.6.1 Controlling 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 following init script arguments to `service cmd`. The `cmdaemonctl` command also works with some arguments:

`service cmd` **Arguments**	**Description**
`stop`	stop the cluster management daemon
`start`	start the cluster management daemon
`reload`	reload configuration of the cluster management daemon
`force-reload`	force reload configuration of the cluster management daemon
`restart`	restart the cluster management daemon
`try-restart`	try to restart the cluster management daemon, but only if it is running
`status`	report whether cluster management daemon is running
`full-status`*	report detailed statistics about cluster management daemon
`upgrade`*	update database schema after version upgrade (*expert only*)
`debugon`*	enable debug logging (*expert only*)
`debugoff`*	disable debug logging (*expert only*)
`logconf`*	Reload log configuration

* arguments that work with `cmdeamonctl` as well as with `service cmd`

Example

Restarting the cluster management daemon on the head node of a cluster:

```
[root@mycluster ~]# service cmd restart
Redirecting to /bin/systemctl restart  cmd.service
[root@mycluster ~]#
```

Example

Viewing the resources used by CMDaemon, and some other useful information:

```
[root@rima92 etc]# service cmd full-status
   CMDaemon version 2.1 is running (active)
   Running locally

   Current Time: Fri, 29 Jan 2021 01:48:28 CET
   Startup Time: Thu, 28 Jan 2021 15:45:17 CET
   Uptime: 10h 3m

   CPU Usage: 66.8112u 50.5393s (0.3%)
   Memory Usage: 172MB

   Sessions Since Startup: 29
   Active Sessions: 7

   Number of occupied worker-threads: 7
   Number of free worker-threads: 14

   Connections handled: 2397
   Requests processed: 6850
   Total read: 1.98MB
   Total written: 170MB

   Average request rate: 11.4requests/m
   Average bandwidth usage: 4KB/s
```

Example

Restarting the cluster management daemon on a sequence of regular nodes, `node001` to `node040`, of a cluster:

```
[root@mycluster ~]# pdsh -w node00[1-9],node0[1-3][0-9],node040 service cmd restart
```

This uses `pdsh`, the parallel shell command (section 9.1).

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 Chapter7.

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

```
[rima92->device]% cmdaemonversions
Hostname         Version index Version hash
---------------- ------------- ------------
rima92          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 which gathers together versions with the same option:

```
[rima92->device]% cmdaemonversions --join
Version index Version hash Count        Hostnames
------------- ------------ ------------ -------------------------
146,965       e6f593b676   3            rima92,node001..node002
```

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@rima92 ~]# 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@rima92 ~]# 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), `CEPH` (for Ceph), 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@rima92 etc]# service cmd restart
```

or by reloading the logging configuration:

Example

```
[root@rima92 etc]# service cmd logconf
```

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 523) 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 3.15.3), and one of various possible configuration techniques (section 3.15.1).

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

While the cluster manager 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 the cluster manager 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 the cluster manager for LDAP and PAM.

In such a case the configuration file generated by the cluster manager 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 cluster manager 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, A.2 and A.4), are given in Appendix A.5.

2.6.7 CMDaemon Lite

Introduction

As an alternative to the regular CMDaemon, the cluster manager 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 3.6 or higher.

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

A zipped package can be picked up on the head node from the cluster manager repositories with:

Example

```
yum install cm-lite-daemon
```

This places a zip file at `/cm/shared/apps/cm-lite-daemon-dist/cm-lite-daemon.zip`. This file should be moved to and unzipped on the *lite node*. The lite node is the machine that is to run `cm-lite-daemon`.

```
[root@rima92 ~]# scp  /cm/shared/apps/cm-lite-daemon-dist/cm-lite-daemon.zip \
 root@lite01:/opt/cm-lite-daemon.zip
[root@rima92 ~]# ssh root@lite01
[root@lite01 ~]# cd /opt
[root@lite01 opt]# unzip cm-lite-daemon.zip
[root@lite01 opt]# cd cm-lite-daemon
[root@lite01 cm-lite-daemon]# ls -1l --group-directories-first
total 44
drwxr-xr-x 7 root root 4096 Nov 11 13:22 cm_lite_daemon
drwxr-xr-x 2 root root  104 Nov 11 13:23 etc
drwxr-xr-x 2 root root   76 Oct 19 16:43 examples
drwxr-xr-x 2 root root    6 Oct 19 16:43 log
drwxr-xr-x 2 root root   78 Oct 19 16:43 service
-rwxr-xr-x 1 root root 4986 Oct 19 16:43 cm-lite-daemon
-rwxr-xr-x 1 root root  740 Oct 19 16:43 cm-lite-daemon_ctl
-rwxr-xr-x 1 root root 2469 Oct 19 16:43 connection_test
-rwxr-xr-x 1 root root  445 Oct 19 16:43 install-required-pip-packages
-rwxr-xr-x 1 root root  245 Oct 19 16:43 install-required-pip-packages.bat
-rwxr-xr-x 1 root root 5401 Oct 19 16:43 register_node
-rwxr-xr-x 1 root root 2808 Oct 19 16:43 request_certificate
-rwxr-xr-x 1 root root 3907 Oct 19 16:43 unregister_node
```

The lite node needs a certificate, and to be registered before `cm-lite-daemon` can run on it. The easiest way to do this is to use the `register_node` utility which is one of the unzipped files in the preceding list. Running it:

- installs required Python packages
- requests a new certificate
- registers the lite node with the head node
- installs `cm-lite-daemon` as a service.

After `register_node` is run, CMDaemon running on the head node is able to see the certificate request. Depending on the network that the CMDaemon Lite on the lite node is connected to, the certificate will be automatically issued, just like it is for regular RIMA nodes being installed. However if CMDaemon Lite is connected via different network, then the certificate must be issued manually, which can be done as follows:

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

Example

```
[rima92->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:

```
rima92->cert]% issuecertificate --days 10000 1
```

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

On a Linux machine `register_node` starts `cm-lite-daemon` as a service, so that the following commands work as expected:

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

On non-Linux operating systems, `cm-lite-daemon` must be started manually.

CMDaemon Lite can be tested by first running it in a foreground shell environment:

Example

```
[root@lite01 cm-lite-daemon]# ./cm-lite-daemon
```

The lite node should then show up as being in the UP state in RIMA View or `cmsh`.

Afterwards the `cm-lite-daemon` Python script can be registered to be autostarted. The administrator should ensure that the running directory for this is set correctly.

The `cm-lite-daemon` service can alternatively simply be run as a foreground process when needed.

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 81) 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 the cluster manager, and so —perhaps rather obviously—a cluster manager 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 ~]# service named restart
```

3

Configuring The Cluster

After the RIMA Cluster 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 7). 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 and Omni-Path are 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 explains how disk layouts are configured, as well as how diskless nodes are set up.

Section 3.10 describes how NFS volumes are exported from an NFS server and mounted to nodes using the integrated approach of the cluster manager.

Section 3.11 describes how services can be run from the cluster manager.

Section 3.12 describes how a rack can be configured and managed with the cluster manager.

Section 3.13 describes how GPUs can be configured with the cluster manager.

Section 3.14 describes how custom scripts can replace some of the default scripts in use.

Section 3.15 discusses configuration alternatives that are not based on CMDaemon.

Section 3.16 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 RIMA View—can be used to carry out cluster management (Chapter 2), the RIMA 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 RIMA 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 RIMA View front end relies on the clickpath: `Cluster`→`Settings`.

This section now continues with the RIMA View description.
The clickpath: `Cluster`→`Settings` brings up RIMA View's cluster `Settings` window (figure 3.1):

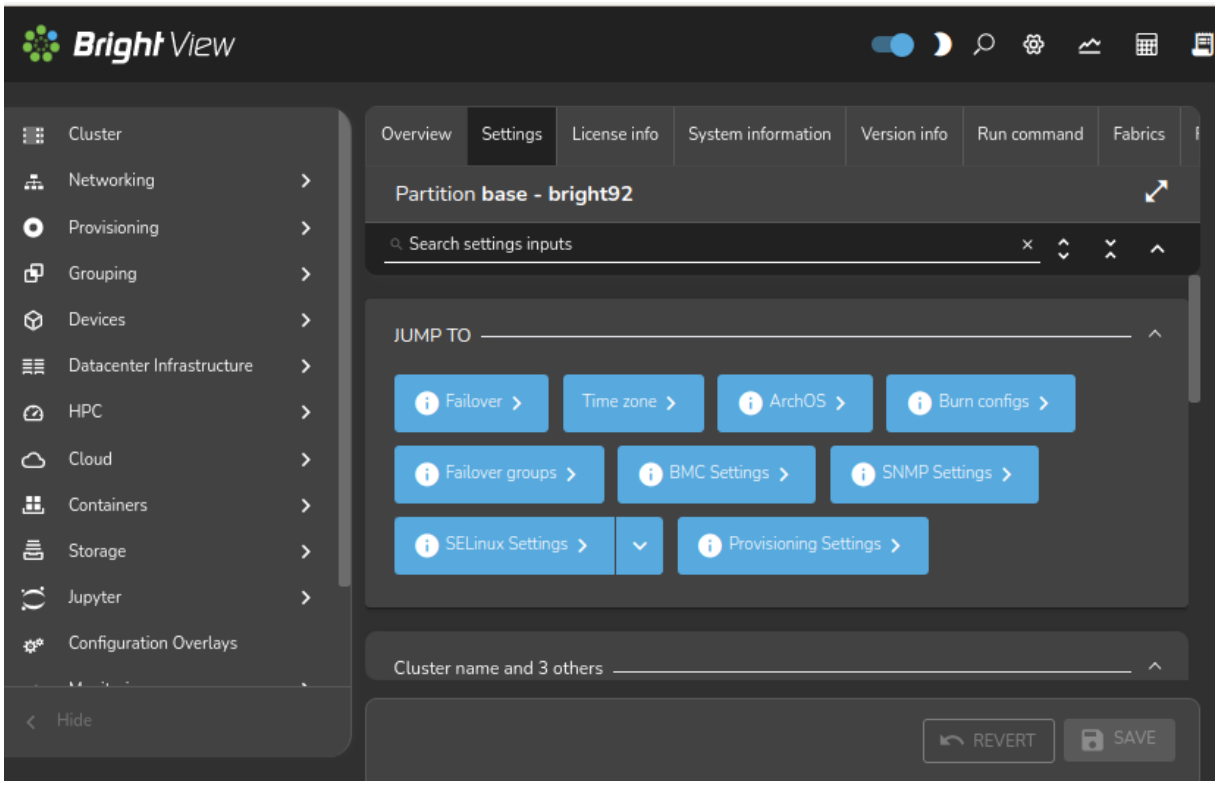


Figure 3.1: Cluster Settings

The navigational overview (figure 2.4) indicates how the main cluster `Settings` window fits into the organizational layout of the cluster manager.

The cluster `Settings` window allows changes to be made to many of the global cluster settings. Its values can in some cases be overridden by more specific configuration levels, such as category-level or node-level configuration. The main cluster settings in figure 3.1 are related to the cluster name-related settings, cluster networking, and some miscellaneous global cluster settings.

3.1.1 Cluster Configuration: Various Name-Related Settings

In the RIMA View `Settings` window, the following defaults can be viewed and modified for names throughout the cluster.

- `Cluster name`: (default name: `RIMA 9.2-stable Cluster`)
- `External network`: (default name: `externalnet`)
- `Internal network`: (default name: `internalnet`)
- `Default category`: (default name: `default`)
- How the nodes of the cluster are named:
  - `Node name`: the base prefix, also called *basename* (default prefix name: `node`)
  - `Node digits size`: number of digits in suffix of node name (default size: 3)

  The global node naming structure can be managed in RIMA View via the clickpath `Cluster`→`Settings`→`NODE BASENAME`. It can also be managed in `cmsh` via the parameters `nodebasename` and `nodedigits`, under the `partition` mode (page 79) of `cmsh`.

Changing the naming and digit size only affects nodes created after the setting.

Renaming of existing node names is possible using the `rename` command from `device` mode of `cmsh`, (section 2.5.5, page 49).

Cloning of nodes is also possible, and can save some work for the cluster administrator, due to some heuristics used to get the cloned node configured correctly (page 44).

3.1.2 Cluster Configuration: Some Network-Related Settings

These 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 RIMA 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 RIMA View, then the changes to the `nameserver` and `searchdomain` values can instead be carried out via `partition` mode (page 77).

Limit to the number of search domains: In older versions of the Linux operating system, the number of names that can be set as search domains used by the cluster has a maximum limit of 6 by default, with a total of 256 characters.

More recent versions of glibc—from glibc 2.26 onward—no longer set a limit.

Because using more than 6 search domains is unsupported by older glibcs, some administrators take the risk of forcefully installing a newer glibc, overriding the official repository dependency restrictions. This results in a system that is unsupported by the distribution, and is also unsupported by the cluster manager.

Instead of trying to set more than the officially supported number of search domains, the use of FQDNs is advised as a workaround.

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

```
[rima92]% network use ibnet
[rima92->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

```
[rima92]% partition
[rima92->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.

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 RIMA View, the clickpath 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 RIMA View, the clickpath 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 RIMA View, the time zone parameters can be jumped to via the clickpath:
  `Cluster`→`Settings`→`JUMP TO`→`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

  ```
  [rima92]% partition use base
  [rima92->partition[base]]% set timezone america/los_angeles
  [rima92->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 cluster manager 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 RIMA View via the clickpath:

`Cluster`→`Settings`→`JUMP TO`→`BMC Settings`

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

- `User name`: (default: `rima`)
- `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 the cluster manager runs on sends e-mails for the administrator to the root e-mail address. The administrator e-mail address can be changed within the cluster manager so that these e-mails are received elsewhere.

- In RIMA View, an e-mail address (or space-separated addresses) can be set in the `Administrator e-mail` field via the clickpath `Cluster`→`Settings`→`CLUSTER NAME` (figure 3.1).
- In `cmsh`, the e-mail address (or space-separated addresses) can be set in `partition` mode, using the `base` object as follows:

  Example

```
[rima92]% partition use base
[rima92->partition[base]]% set administratore-mail alf@example.com beth@example.com
[rima92->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 RIMA View: the `Relay Host` field 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@rima92 ~]# cmsh
[rima92]% partition use base
[rima92-> partition[base]]% set relayhost mail.example.com
[rima92-> 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 in `partition` mode, then a node reboot deploys the change for the node. Setting the `AdvancedConfig` (page 528) `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 the cluster manager's `testemail` command. The command is run from within the `main` mode of `cmsh`. It sends a test e-mail out using CMDaemon:

```
[root@rima92 ~]# 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@rima92 ~]# cmsh -c "main; testemail"
Mail delivered to postfix
You have new mail in /var/spool/mail/root
[root@rima92 ~]# 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 the high availability (HA) feature of the cluster for head nodes, the administrator can click on the `Failover` option in RIMA View. This opens up a subwindow that can be used to configure HA-related parameters (section 10.4.6).

Failover Groups Settings

To access the high availability feature of the cluster for groups of regular nodes, the administrator can click on the `Failover groups` option in RIMA View. This opens up a subwindow that can be used to configure failover-groups-related parameters (section 10.5).

Burn Configs

Burning nodes is covered in Chapter 8 of the *Installation Manual*. Burn configuration settings for the cluster can be accessed in RIMA View via the following clickpaths:

- `Cluster[Partition base]`→`Settings`→`Default burn configuration`

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

- `Cluster[Partition base]`→`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 stick to some security standards. This is known as FIPS compliance.

Prior to RIMA Cluster Manager version 9.2, FIPS configuration was configured outside of the cluster manager by the cluster administrator directly on the head and regular nodes. Starting with RIMA Cluster Manager version 9.2, 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 by 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 by 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 the cluster manager, 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(5) sysctl.conf. The cluster manager 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 RIMA Cluster Manager for versions 8.2 and beyond 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 RIMA 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 cluster manager 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 the cluster manager 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(5) limits.conf.

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 in an operating system that is not managed by the cluster manager. However, it is set to 131072 by default for a system managed by the cluster manager. The nofile value is defined by the cluster manager 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 the cluster manager 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 78 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 74 under the subheading "Changing External Network Parameters For The Cluster". This is the only network for which the cluster manager 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 81 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.

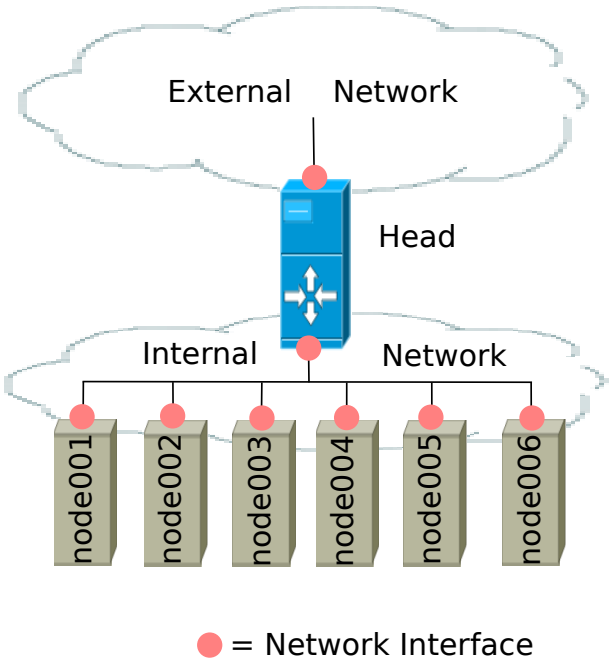


Figure 3.2: Network Settings Concepts

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 75 under the subheading "The IP address of the cluster".
- Changing the configuration of the internal network interfaces is described on page 79 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 244.
- Changing the configuration of `globalnet` is described on page 81 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 RIMA View, a network can be configured via the clickpath `Networking`→`Networks`, which opens up the `Network list` subwindow (figure 3.3):

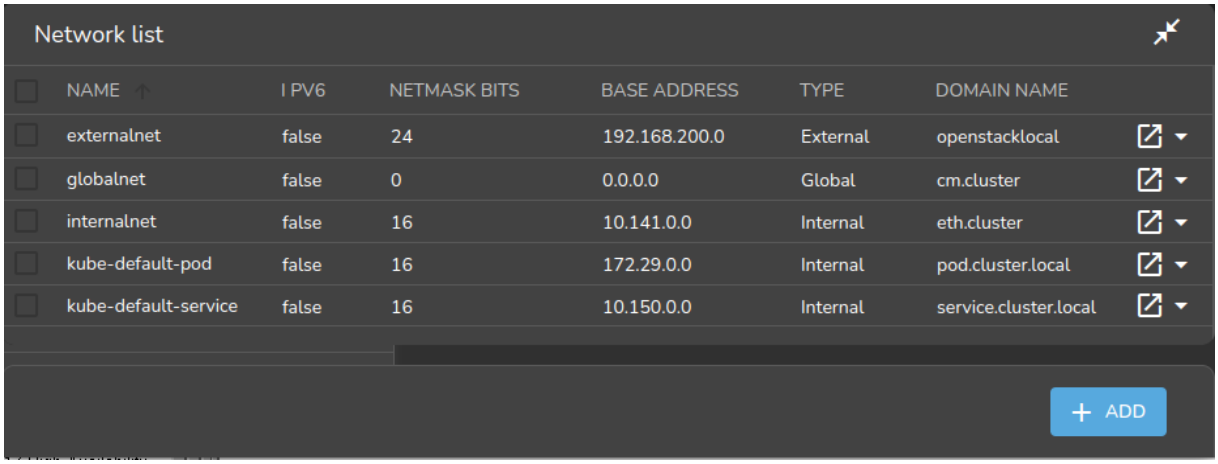


Figure 3.3: Networks

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).

Selecting a network such as `internalnet` or `externalnet` in the list of networks, for example by double-clicking the row, opens up a new `Overview` window for that network. Some resizing of the windows is usually needed to view the new window properly on a standard screen.

The `Overview` window by default lists the device names and device properties that are in the network. For example the `internal` network typically has some nodes, switches, and other devices attached to it, each with their IP addresses and interface (figure 3.4).

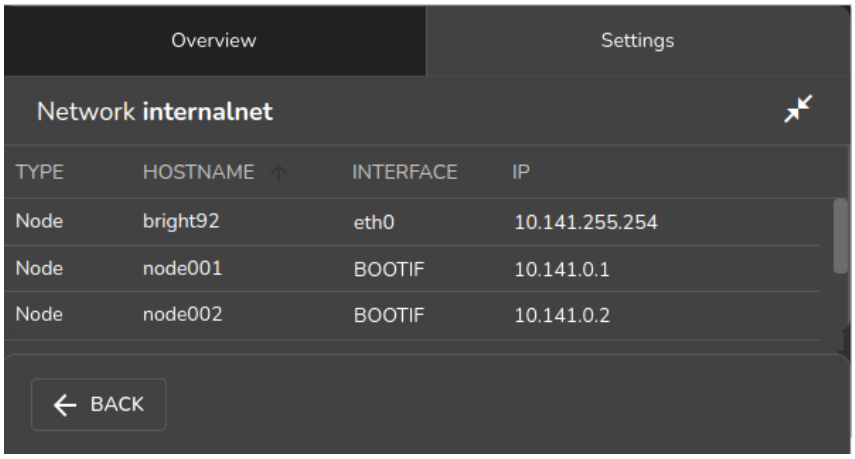


Figure 3.4: Network Overview

Selecting the `Settings` option opens a scrollable pane that allows a number of network properties to be changed (figure 3.5).

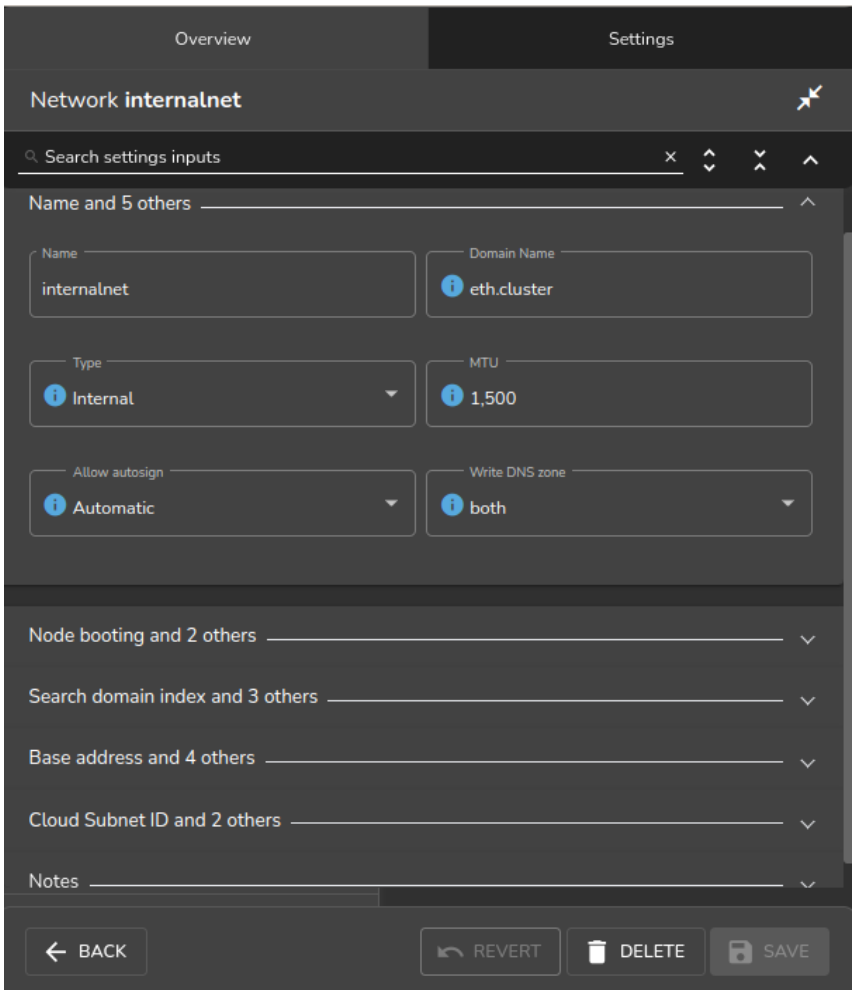


Figure 3.5: Network Settings

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.
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 RIMA View via the Networking resource, selecting a network, and then setting Node booting from within the network settings. • It can be set in cmsh via the network mode, selecting a network, and then setting nodebooting.
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 the cluster manager has no MAC addresses associated with the node. • It can be set in RIMA 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.
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*).
Broadcast address	Broadcast address of the network.
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

Table 3.1: *Network Configuration Settings*

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 installed on the head node is a useful tool for calculating or checking such IP subnet values (`man sipcalc` or `sipcalc -h` for help on this utility):

Example

```
user@rima92:~$ 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
```

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.

3.2.2 Adding Networks

Once a network has been added, it can be used in the configuration of network interfaces for devices.

In RIMA 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 RIMA View, via the `Settings` window of the cluster object (figure 3.1).

In `cmsh` the assignment to `Management network` and `External network` can be set or modified from the `base` object in `partition` mode:

Example

```
[root@rima92 ~]# cmsh
[rima92]% partition use base
[rima92->partition[base]]% set managementnetwork internalnet; commit
[rima92->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 RIMA View, the clickpath for modifying the host name is `Devices`→`Head Nodes`→`Edit`→`Settings`→`Hostname`→`Save`. That is, under the `Devices` resource, the `Head Nodes` option is selected. The `Edit` button can then be used to edit the host. This opens up a pane, and the `Settings` option can then be selected. The `Hostname` record can then be modified (figure 3.6), and saved by clicking on the `Save` button. When setting a hostname, a domain is not included.

  After the change, as suggested by RIMA View, the head node must be rebooted.

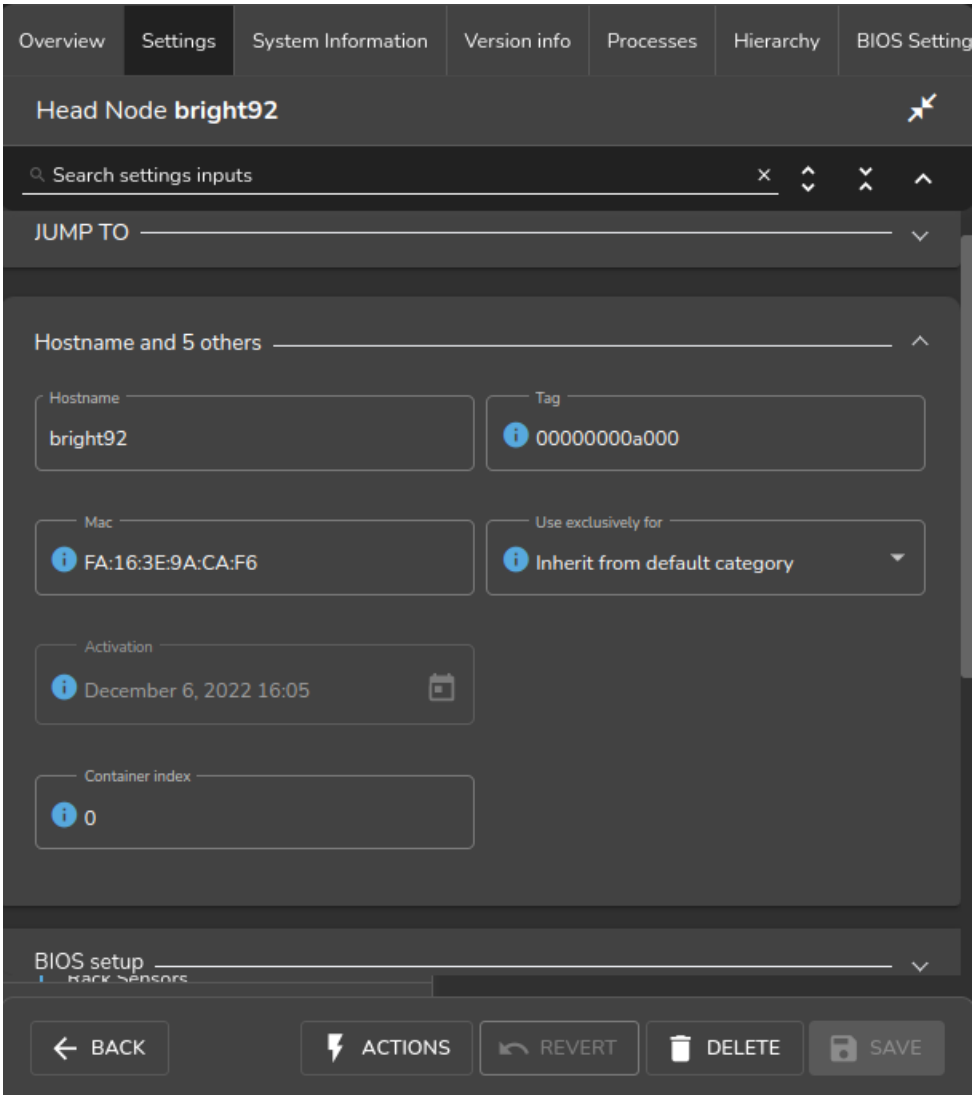


Figure 3.6: Head Node Settings

- In `cmsh`, the hostname of the head node can also be changed, via device mode:

  Example

```
[root@rima92 ~]# cmsh
[rima92]% device use rima92
[rima92->device[rima92]]% 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@rima92 ~]# sleep 30; hostname -f foobar.cm.cluster
[root@rima92 ~]#
```

  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 the cluster manager 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

```
[rima92]% device use rima92
[rima92->device[rima92]]% interfaces
[rima92->device[rima92]->interfaces]% use eth1
[rima92->device[rima92]->interfaces[eth1]]% set additionalhostnames test
[rima92->device*[rima92*]->interfaces*[eth1*]]% commit
[rima92->device[rima92]->interfaces[eth1]]%
Fri Oct 12 16:48:47 2012 [notice] rima92: Service named was restarted
[rima92->device[rima92]->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 3.11.1) of changes made to service configurations by RIMA 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 RIMA 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 is not to be confused with the IP address of the internal network (page 70).
     - 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 the Settings options of the cluster object resource in RIMA View (figure 3.1):

   - 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.

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 the cluster manager 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 the cluster manager, 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 RIMA View or cmsh:

With RIMA View: The cluster network object and associated settings are accessed as follows:

The external network object:
The external network object is accessed via the click path
Networking→Networks→externalnet→Edit→Settings
which reaches the window shown in figure 3.7:

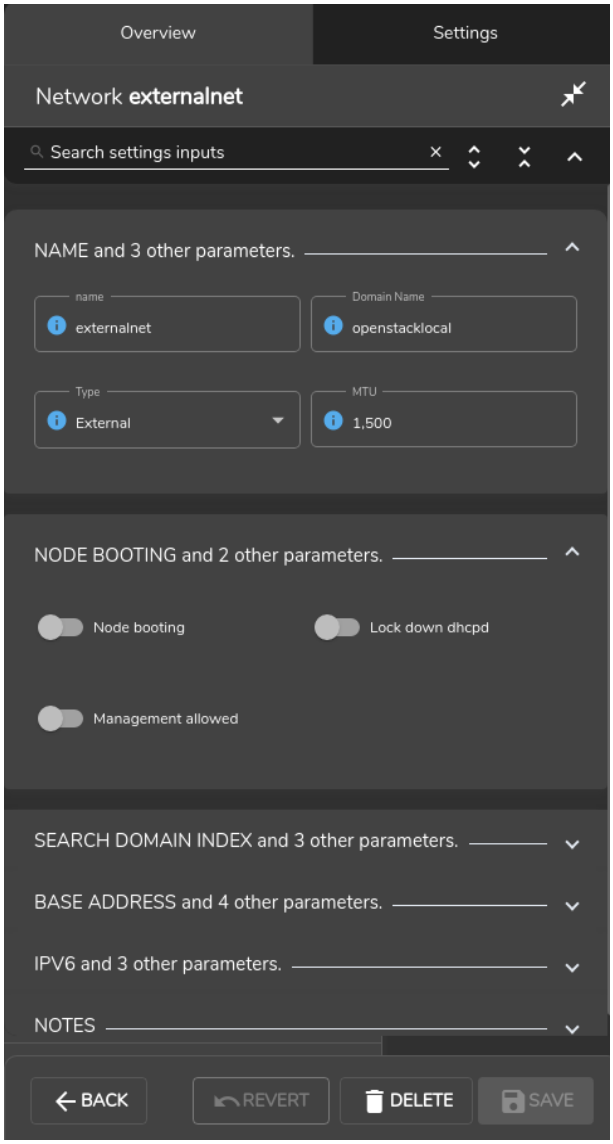


Figure 3.7: Network Settings For External Network

The cluster object:
The cluster object and associated settings are accessed as shown in figure 3.1

The head node IP address:
For a head node `rima92`, where the IP address is used by the interface `eth0`, the static IP address can be set via the clickpath `Devices`→`Head Nodes`→`Edit[rima92]`→`Settings`→`JUMP TO`→`Interfaces`→`Edit[eth0]`→`IP` (figure 3.8).

With `cmsh`: The preceding RIMA View configuration can also be done in `cmsh`, using the `network`, `partition` and `device` modes, as in the following example:

Example

```
[rima92]% network use externalnet
[rima92->network[externalnet]]% set baseaddress 192.168.1.0
[rima92->network*[externalnet*]]% set netmaskbits 24
[rima92->network*[externalnet*]]% set gateway 192.168.1.1
[rima92->network*[externalnet*]]% commit
```

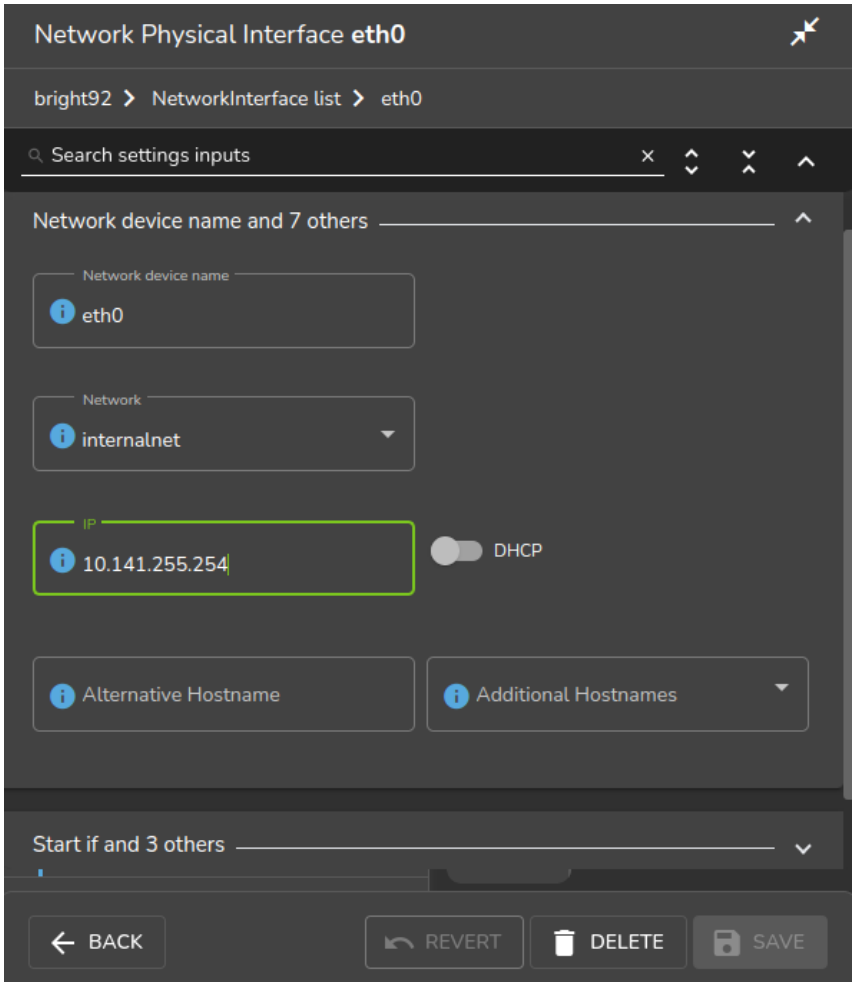


Figure 3.8: Setting The IP Address On A Head Node In RIMA View

```
[rima92->network[externalnet]]% partition use base
[rima92->partition[base]]% set nameservers 192.168.1.1
[rima92->partition*[base*]]% set searchdomains x.com y.com
[rima92->partition*[base*]]% append timeservers ntp.x.com
[rima92->partition*[base*]]% commit
[rima92->partition[base]]% device use rima92
[rima92->device[rima92]]% interfaces
[rima92->device[rima92]->interfaces]% use eth1
[rima92->device[rima92]->interfaces[eth1]]% set ip 192.168.1.176
[rima92->device[rima92]->interfaces*[eth1*]]% commit
[rima92->device[rima92]->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 RIMA 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:

```
[rima92->device[rima92]->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 3.9 in Chapter 3 of the *Installation Manual*). The time servers can be changed using RIMA View as in 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 78)
2. the cluster network for the internal network (page 79)
3. the individual device network interface (page 79)
4. the node categories network-related items for the device (page 81), 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 RIMA 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 or OmniPath 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 RIMA 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 81). 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 RIMA View, the network for a node, such as a physical node `node001`, can be configured via a clickpath `Devices`→`Nodes[node001]`→`Interface`. 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 79 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

```
[rima92] device interfaces node001
[rima92->device[node001]->interfaces]% add physical eth1
[rima92->...->interfaces*[eth1*]]% set network externalnet
[rima92->...->interfaces*[eth1*]]% set additionalhostnames extra01
[rima92->...->interfaces*[eth1*]% set ip 10.141.1.1
[rima92->...->interfaces*[eth1*]]% commit
[rima92->...->interfaces[eth1]]% ..
[rima92->...->interfaces]% ..
[rima92->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 RIMA View, using the `Settings` option for `Node categories`. This is in the clickpath `Grouping`→`Node categories[default-image]`→`Settings`
- or via the `category` mode in `cmsh`, for the `default` node category.

If there are individual node settings that have been configured in RIMA 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 RIMA 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 78, 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 the cluster manager 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 F.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 the cluster manager. It is required in the cluster manager 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 7.6.1 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:

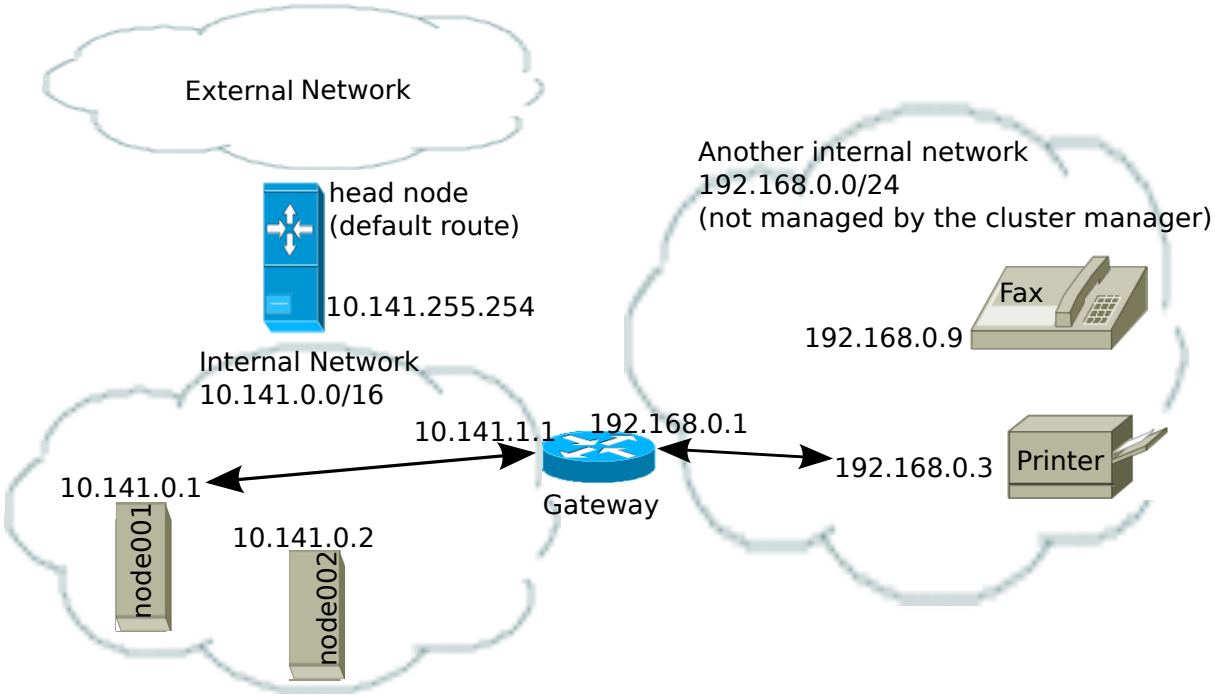


Figure 3.9: Example Of A Static Route To A Printer On Another Subnet

then an example session for setting the static route is as follows:

Example

```
[root@rima92 ~]# cmsh
[rima92 ]% device use node001
[rima92->device[node001]]%
[rima92->device[node001]]% staticroutes
[rima92->device[node001]->staticroutes]% list
Name (key)                     Gateway          Destination
------------------------------ ---------------- --------------
[rima92->device[node001]->staticroutes]% add printerroute
[rima92->...*[printerroute*]]% set gateway 10.141.1.1
[rima92->...*[printerroute*]]% set destination 192.168.0.3
[rima92->...*[printerroute*]]% set networkdevicename bootif
[rima92->...*[printerroute*]]% commit
[rima92->...[printerroute]]% show
Parameter                      Value
------------------------------ ------------------------------
Destination                    192.168.0.3/32
Gateway                        10.141.1.1
Name                           printerroute
Network Device Name            bootif
```

```
Revision
[rima92->device[node001]->staticroutes[printerroute]]% exit
[rima92->device[node001]->staticroutes]% list
Name (key)                      Gateway          Destination
------------------------------- ---------------- --------------
printerroute                    10.141.1.1       192.168.0.3/32
[rima92->device[node001]->staticroutes]% exit; exit
[rima92->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@rima92 ~]# 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@rima92->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).

3.2.4 Tools For Viewing Cluster Connections

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@rima92 ~]# cmsh
[rima92]% device
[rima92->device]% routes
Node       Destination      Gateway          Netmask          Interface
---------- ---------------- ---------------- ---------------- ---------
rima92   0.0.0.0          192.168.200.254  0.0.0.0          eth1
rima92   10.141.0.0       0.0.0.0          255.255.0.0      eth0
rima92   169.254.0.0      0.0.0.0          255.255.0.0      eth1
rima92   169.254.169.254  192.168.200.254  255.255.255.255  eth1
rima92   192.168.200.0    0.0.0.0          255.255.255.0    eth1
node001    0.0.0.0          10.141.255.254   0.0.0.0          eth0
node001    10.141.0.0       0.0.0.0          255.255.0.0      eth0
node001    169.254.0.0      0.0.0.0          255.255.0.0      eth0
```

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 43, 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, for unexpected gateways, when the expected gateway is <*expected-gateway*>, the following command may be used:

Example

```
[root@rima92 ~]# cmsh
[rima92]% device
[rima92->device]% routes --category default | awk 'if ($3 != "<expected-gateway>") print $0'
...
```

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

```
[rima92->device]% connections
Node      Type     Source   Port  Destination  Port  State
--------- -------- -------- ----- ------------ ----- ----------
rima92  TCP      0.0.0.0  22    0.0.0.0      0     Listening
rima92  TCP      0.0.0.0  25    0.0.0.0      0     Listening
rima92  TCP      0.0.0.0  111   0.0.0.0      0     Listening
rima92  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

```
[rima92->device]% connections | head -2; connections | grep Listen | grep " 53"
Node      Type     Source             Port     Destination     Port     State
--------- -------- ------------------ -------- --------------- -------- -----------
```

```
rima92  TCP      10.141.255.254     53       0.0.0.0         0        Listening
rima92  TCP      127.0.0.1          53       0.0.0.0         0        Listening
rima92  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

```
[rima92->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 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

```
[rima92->device]% connectivity
Source    Destination Result       ID   Latency
--------- ----------- ------------ ---- ------------
rima92  node001     Ok           0    0.4ms
rima92  node002     Ok           0    0.3ms
rima92  node003     Ok           0    0.4ms
rima92  rima92    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

```
[rima92->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.

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

```
[rima92->device[node001]->interfaces]% add bridge br0
[rima92->...->interfaces*[br0*]]% set network internalnet
[rima92->...->interfaces*[br0*]]% set ip 10.141.1.1
[rima92->...->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

```
[rima92->...->interfaces*[br0*]]% set interfaces eth1 eth2; exit
[rima92->...->interfaces*]% clear eth1 ip; clear eth1 network
[rima92->...->interfaces*]% clear eth2 ip; clear eth2 network
[rima92->...->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 the cluster manager using `cmsh` or RIMA 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@rima92 ~]# cmsh
[rima92]% device interfaces node001
[rima92->device[node001]->interfaces]% list
Type         Network device name  IP              Network
------------ -------------------- --------------- ------------
physical     BOOTIF [prov]        10.141.0.1      internalnet
[rima92->device[node001]->interfaces]% list -h | tail -4
Arguments:
   type
     alias, bmc, bond, bridge, netmap, physical, tunnel, vlan

[rima92->device[node001]->interfaces]% add vlan BOOTIF.1
[rima92->device*[node001*]->interfaces*[BOOTIF.1*]]% commit
[rima92->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
[rima92->...[BOOTIF.1]]% set network internalnet
[rima92->...[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 RIMA View

Within RIMA View a VLAN interface can be added for a node such as `node001` via the clickpath: `Devices`→`Nodes`→`Edit[node001]`→`Settings`→`JUMP TO`→`Interfaces`→`+ADD[Network VLAN interface]` (figure 3.10).

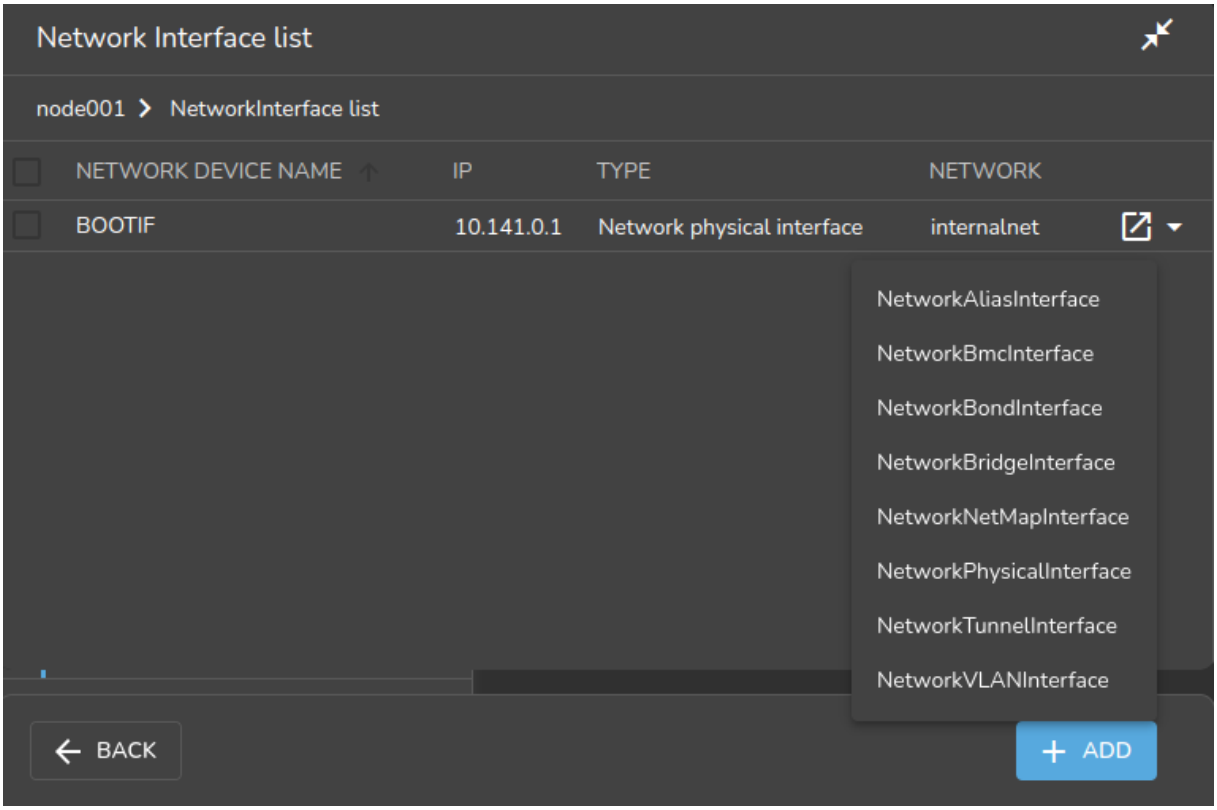


Figure 3.10: Configuring a VLAN via RIMA View

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 80) 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

```
[rima92->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 RIMA View or `cmsh`, the bonding mode is just another bonding option specified as part of the options string. However in the cluster manager 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 86.
- 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 10), 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 10.1) 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 10.1) 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 221.

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

The cluster manager 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 depending on the bond mode, either interface may be used for PXE booting. Each interface has its own MAC address, which means that if the interface MAC address does not match the node MAC address, then the node loops in the node-installer waiting to be identified as a new node (section 5.4.2).

To have the node-installer recognize any interface MAC in the bonded interface for a node, the following steps should be carried out.

- A `GlobalConfig` directive (page 529):

  ```
  GlobalConfig={ "DeviceAnyMAC=1" }
  ```

  should be set in `cmd.conf` on the head nodes.

- The MAC attribute should be set for each of the interfaces used by the bond interface. For example, if `node001` is using the component interfaces `eth0` and `eth1` for a bond interface called `bond0`, then the MAC address for each of the component interfaces can be set

  Example

  ```
  [rima92]% device use node001; interfaces
  [rima92->device[node001]->interfaces]% set eth0 mac <First interface MAC address>
  [rima92->device*[node001*]->interfaces*]% set eth1 mac <Other interface MAC address>
  [rima92->device*[node001*]->interfaces*]% commit
  ```

- The MAC property for the node should be set to an interface MAC address. It does not matter which one is matched.

  Example

  ```
  [rima92]% device use node001
  [rima92->device[node001]]% set mac <Any interface MAC address>; commit
  ```

After these steps have been carried out, what happens during boot is that the node-installer not only also looks at the MAC address for the node, but it also looks at the MAC addresses on the interfaces, and automatically updates the MAC property for the node to whichever NIC was used to PXE boot.

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 86.

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 And Omni-Path Interfaces

On clusters with an InfiniBand interconnect, the InfiniBand Host Channel Adapter (HCA) in each node must be configured before it can be used. A similar requirement applies for Omni-Path and its fabric controller. The cluster manager manages InfiniBand and Omni-Path configuration in the same way. Therefore Omni-Path management is usually also implied whenever InfiniBand is discussed in the cluster manager manuals.

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 3.10.5.

3.6.1 Installing Software Packages

On a standard RIMA Cluster 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, and iWarp protocol hardware such as the hardware-accelerated RDMA over ethernet provided by Intel.

By default, all relevant OFED packages are installed on the head node and software images. It is possible to replace the distribution OFED with an OFED provided by the cluster manager repository or another custom version. 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 the cluster manager OFED and Omni-Path packages is covered in section 7.6 and section 7.7 respectively 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 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 the cluster manager, an administrator sets a subnet manager service[1] to start at boot-time with a command such as:

```
systemctl enable opensm.service
```

However, for clusters managed by the cluster manager, 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@rima92 ~]# cmsh -c "device roles <node>; assign subnetmanager; commit"
   ```

[1] usually `opensm`, but `opensmd` in SLES

where <node> is the name of a node on which it will run, for example: rima92, node001, node002...

In RIMA 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 clickpath for this, for a node node002 for example, is Devices→Nodesnode002→Settings→Roles→Add[Subnet manager role].

2. by setting the service up. Services are covered more generally in section 3.11.

   In cmsh this is done with:

   Example

```
[root@rima92 ~]# cmsh
[rima92]% device services node001
[rima92->device[node001]->services]% add opensm
[rima92->device[node001]->services*[opensm*]]% set autostart yes
[rima92->device[node001]->services*[opensm*]]% set monitored yes
[rima92->device[node001]->services*[opensm*]]% commit
[rima92->device[node001]->services[opensm]]%
```

   In RIMA 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 clickpath for this, for a node node002 for example, is: Devices→Nodesnode002→Settings→Services[*Service*].

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.

3.6.3 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

```
[rima92->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@rima92 ~]# 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@rima92 ~]# 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:

```
[rima92->device[rima92]->interfaces]% add physical ib0
[rima92->device[rima92]->interfaces*[ib0*]]% set network ibnet
[rima92->device[rima92]->interfaces*[ib0*]]% set ip 10.149.255.254
[rima92->device[rima92]->interfaces*[ib0*]]% commit
```

As with any change to the network setup, the head node needs to be restarted to make the above change active.

3.6.4 Enabling SNMP

Some InfiniBand switches support SNMP.

The cluster manager can be used to manage such an InfiniBand switch. This requires explicitly enabling SNMP, by changing the default `yes` setting for `disablesnmp` to `no`:

Example

```
[rima92]% device add ibswitch myibswitch
[rima92->device[myibswitch]]% set disablesnmp no; commit
```

Configuration of SNMP settings for switches is described in further detail in section 3.8.2.

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@rima92 ~]# su - cmsupport
[cmsupport@rima92 ~]$ cd /cm/shared/apps/imb/current/
[cmsupport@rima92 current]$ ./setup.sh
[cmsupport@rima92 current]$ cd ~/BenchMarks/imb/2017
[cmsupport@rima92 2017]$ module load openmpi/gcc
[cmsupport@rima92 2017]$ module initadd openmpi/gcc
[cmsupport@rima92 2017]$ make -f make_mpi2
[cmsupport@rima92 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

The cluster manager 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 3.16 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 the cluster manager.

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:

```
[rima92 ~]# 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`:

```
[rima92 ~]# 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@rima92 ~]# cmsh
[rima92]% device
[rima92->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

[rima92->device]% addinterface -n node001..node150 bmc ipmi0 bmcnet 10.148.0.1
[rima92->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:

```
[rima92->device[rima92]->interfaces]% add alias eth0:0
[rima92->device[rima92]->interfaces*[eth0:0*]]% set network bmcnet
[rima92->device[rima92]->interfaces*[eth0:0*]]% set ip 10.148.255.254
[rima92->device[rima92]->interfaces*[eth0:0*]]% commit
[rima92->device[rima92]->interfaces[eth0:0]]%
Mon Dec 6 05:45:05 rima92: Reboot required: Interfaces have been modified
[rima92->device[rima92]->interfaces[eth0:0]]% quit
[root@rima92 ~]# service network restart
```

As with any change to the network setup, the head node needs to be restarted to make the above change active, although in this particular case restarting the `network` service would suffice.

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@rima92 ~]# 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 `rima` 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

```
[rima92]% partition use base
[rima92->partition[base]]% bmcsettings
[rima92->partition[base]->bmcsettings]% get username
rima
[rima92->partition[base]->bmcsettings]% get password
Za4ohni1ohMa2zew
[rima92->partition[base]->bmcsettings]% set username bmcadmin
[rima92->partition*[base*]->bmcsettings*]% set password
enter new password: ******
retype new password: ******
[rima92->partition*[base*]->bmcsettings*]% commit
```

In RIMA 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`

BMC configuration on a head node is done directly.
For regular nodes the cluster manager 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`.

3.7.3 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 the cluster manager. 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 the cluster manager only for RHEL7-based and RHEL8-based distributions at the time of writing of this section (March 2023).

If Dell was chosen as the hardware vendor when the cluster manager ISO was created for installation, and chosen as the hardware manufacturer when the head node was configured in the cluster manager 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 cluster manager 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). The cluster manager 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@rima92 ~]# cmsh
[rima92]% partition use base
[rima92->partition[base]]% bmcsettings
[rima92->partition[base]->bmcsettings]% get password
12345
[rima92->partition[base]->bmcsettings]% get username
tom
[rima92->partition[base]->bmcsettings]%
```

Sometimes the `rima` user does not have the right privilege to get the correct values. The `racadm` commands then fail.

The `rima` 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 rima, a userID 4, and the privilege can be set to 511.

3.7.4 Identification With A BMC

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.

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 521) can be used.

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 via 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 RIMA View, this is done via the clickpath Devices→Edit→<*switchname*> to select the switch. If the switch does not already exist, then it can be added via 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@rima92 ~]# cmsh
  [rima92]% device
  [rima92->device]% set switch01 ip 10.141.253.2
  [rima92->device*]% commit
  ```

Managing PDUs in RIMA 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

```
[rima92->device[mypdu]]% set powercontrol apc; commit
```

and in RIMA View with the clickpath:

`Devices→Power Distribution Units→Power Distribution Unit list→Power Distribution Unit→Power control`

If it is not set to `apc`, then the list of PDU ports is ignored by default.

3.8.2 Configuring SNMP

Configuring SNMP Community Strings

In order to allow the cluster management software to communicate with the switch or PDU, SNMP must be enabled on it, and the SNMP community strings should be configured correctly.

For InfiniBand switches, SNMP management must also be enabled explicitly in the cluster manager, as indicated in section 3.6.4.

By default, the SNMP community strings for switches and PDUs are set to `public` and `private` for respectively read and write access. 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

```
[rima92]% device use switch01
[rima92->device[switch01]]% snmpsettings
[rima92->device[switch01]->snmpsettings]% get readstring
public
[rima92->device[switch01]->snmpsettings]% get writestring
private
[rima92->device[switch01]->snmpsettings]% set readstring public2
[rima92->device*[switch01*]->snmpsettings*]% set writestring private2
[rima92->device*[switch01*]->snmpsettings*]% commit
```

Alternatively, these properties can also be set in RIMA View via the clickpath:

`Devices→Switches→Edit→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 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 the cluster manager are:

Example

```
[rima92]% device use switch01
[rima92->device[switch01]]% snmpsettings
[rima92->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

```
[rima92->device*[switch01*]->snmpsettings*]% set authenticationprotocol <TAB><TAB>
md5  sha
[rima92->device[switch01]->snmpsettings]% set authenticationprotocol aes
[rima92->device*[switch01*]->snmpsettings*]% set privacyprotocol <TAB><TAB>
aes  des
[rima92->device*[switch01*]->snmpsettings*]% set privacyprotocol aes
[rima92->device*[switch01*]->snmpsettings*]% commit
[rima92->device[switch01]->snmpsettings]%
```

SNMP Traps

The cluster manager 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@rima92 ~]# cmsh
[rima92]% device use node001
[rima92->device[node001]]% roles
[rima92->device[node001]->roles]% assign snmptrap
[rima92->device[node001]->roles*[snmptrap*]]% show
Parameter                      Value
------------------------------ ------------------------------------------------
Access                         public
Add services                   yes
All administrators             no
Alternative script
Arguments
Event                          yes
Mail                           yes
Name                           snmptrap
```

```
Provisioning associations        <0 internally used>
Recipients
Revision
Sender
Server
Type                             SnmpTrapRole
```

Cumulus Switches Configuration

A Cumulus switch is configured by setting a username and password for it in the SNMP settings, then setting up and initializing a custom script for it.

The username and password: are set using the `revision` field to hold the required values. For example, if the username is <*fred*> and the password is <*1234*>, then it can be set for a particular Cumulus device:

Example

```
[rima92->device]% add switch mycumulus
[rima92->device*[mycumulus*]]% snmpsettings
Created empty snmp settings
[rima92->device*[mycumulus*]->snmpsettings*]% set revision "username=<fred>;password=<1234>"
[rima92->device*[mycumulus*]->snmpsettings*]% commit
[rima92->device[mycumulus]->snmpsettings]% exit
```

If all switches are Cumulus switches, then the setting can be carried out at `partition` level with:

Example

```
[rima92->partition[base]]% snmpsettings
[rima92->partition[base]]->snmpsettings]% set revision "username=<fred>;password=<1234>"
[rima92->partition*[base*]]->snmpsettings*]% commit
[rima92->partition[base]]->snmpsettings]% exit
```

The Cumulus custom script: comes with the cluster manager, and is at `/cm/local/apps/cmd/scripts/cm-cumulus-switch.py`. It can be set up and initialized with:

Example

```
[rima92->device*[mycumulus*]]% set controlscript /cm/local/apps/cmd/scripts/cm-cumulus-switch.py
[rima92->device*[mycumulus*]]% commit
[rima92->device[mycumulus]]% initialize
```

The initialization discovers the devices on the network:

Example

```
[bright92->device[cumulus1]]% switchoverview | head -n 20
Device: cumulus1
State : [   UP   ]
Model : Cumulus-Linux 5.0.1 (Linux Kernel 4.19.205-1+cl5.0.0u1)

Port Assignment:

Port   Name   Status Assigned         Uplink Speed    Detected
------ ------ ------ ---------------- ------ -------- ---------------------------------------
1      swp1   UP                      no     100 Gb/s 00:08:4B:A2:10:37, 00:12:E1:97:C2:73...
...
```

3.8.3 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, RIMA View or `cmsh` are used:

- In RIMA View, the switch is selected, and uplinks can be added via the clickpath `Devices`→`Switches`→`Edit`→`Uplinks` (figure 3.11):

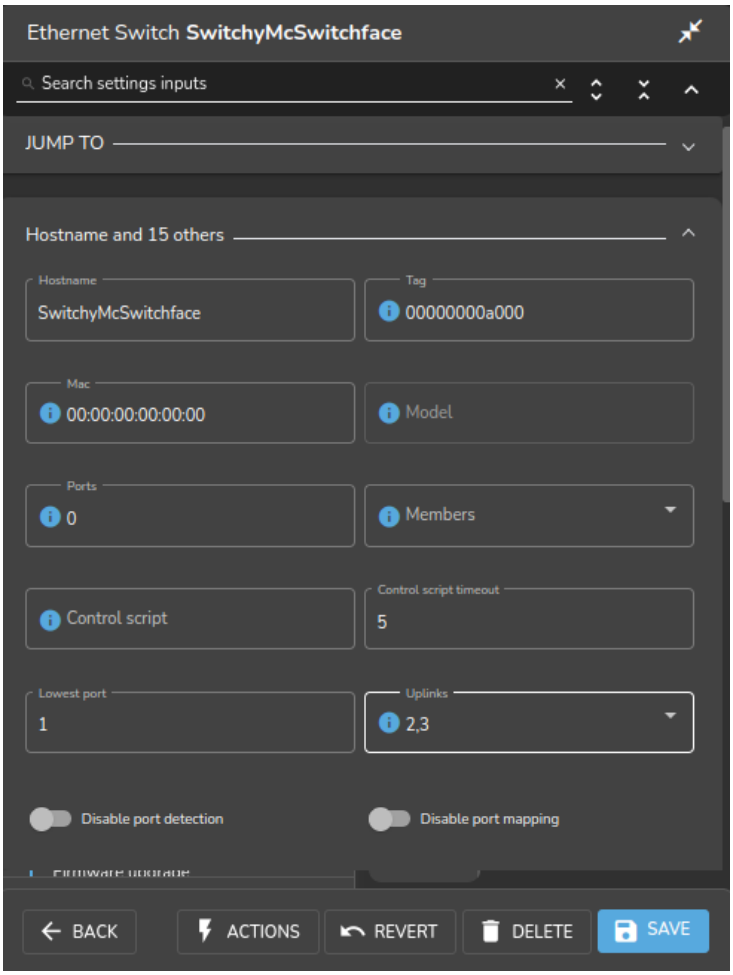


Figure 3.11: Notifying CMDaemon about uplinks with RIMA View

A dialog box then appears, and allows uplink port numbers to be added with a ⊕ button. The state is saved with the `SAVE` 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@rima92 ~]# cmsh
[rima92]% device
[rima92->device]% set switch01 uplinks 15 16
```

```
[rima92->device*]% set switch02 uplinks 01
[rima92->device*]% commit
successfully committed 3 Devices
```

3.8.4 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@rima92 ~]# cmsh
[rima92]% device
[rima92->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:

```
[rima92->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 rima92: 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
```

3.8.5 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@rima92 ~]# cmsh
[rima92]% device use node001
[rima92->device[node001]]% clear switchports
[rima92->device*[node001*]]% commit
[rima92->device[node001]]%
```

Or, for example for the `default` category, with the help of the `foreach` command:

Example

```
[root@rima92 ~]# cmsh
[rima92]% device
[rima92->device]]% foreach -c default (clear switchports); commit
```

3.8.6 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):

```
[rima92->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  -
...
[rima92 ->device]%
```

3.9 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 the cluster manager once the cluster is running. For a head node, however, the disk layout cannot be re-configured after installation by the cluster manager, 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.

3.9.1 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)

3.9.2 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)

3.9.3 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 RIMA View or `cmsh`, as is illustrated by the example in section 3.9.4, which covers changing a node from disked to diskless mode:

3.9.4 Changing A Disk Layout From Disked To Diskless

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 RIMA View or `cmsh` as follows:

Changing A Disk Layout Using RIMA View

To change a disk layout with RIMA 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, the clickpath is
`Devices`→`Nodes`→`Edit`→`Settings`→`Installing`→`Disk setup`
(figure 3.12):

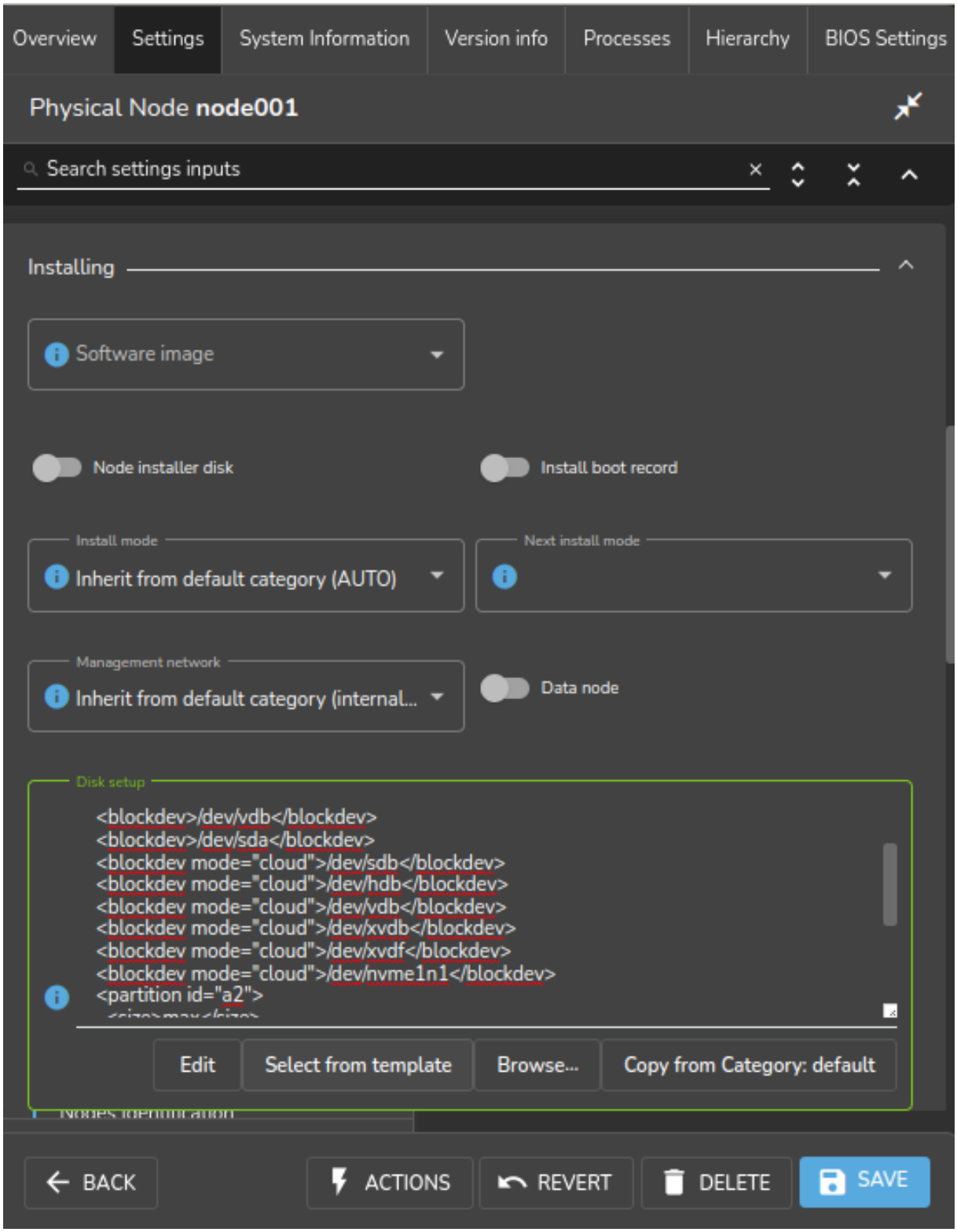


Figure 3.12: Changing a disked node to a diskless node with RIMA View

The `Disk setup` field can then be edited.

Clicking on the `Select from template` button shows several possible ready-made configurations that can be loaded up from the CMDaemon database, and if desired, the copy stored for the node can

be edited to suit the situation.

To switch from the existing disk layout to a diskless one, the diskless XML configuration template is loaded via the `Select from template` button, and saved to the node or node category.

The `Browse` button can be used to upload a custom configuration via the browser, and there is also a `Copy from Category:default` button that can be used to copy the category configuration to the node.

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@rima92 ~]# cmsh
[rima92]% device use node001
[rima92->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

```
[rima92->device[node001]]% set disksetup /cm/local/apps/cmd/\
etc/htdocs/disk-setup/slave-diskless.xml
```

In these preceding RIMA 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 3.10, which covers the configuration of NFS exports and mounts in general.

3.10 Configuring NFS Volume Exports And Mounts

NFS allows unix 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 the cluster manager, 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 7 and 8.

  The settings that determine client module loading are set in the file `rdma.conf` so that the service auto-loads by default.

  - For RHEL7 this file is `/etc/rdma/rdma.conf`
  - For RHEL8 this file is `/etc/rdma/modules/rdma.conf`.

- An alternative to NFS over RDMA for very fast file systems is the massively parallel and free (GPLv2) Lustre filesystem, running over InfiniBand.

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 RIMA View via the "`Services`" option (section 3.11) 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@rima92 ~]# cmsh
[rima92]% device use rima92
[rima92->device[rima92]]% services
[rima92->device[rima92]->services]% add autofs
[rima92->device*[rima92*]->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
[rima92->device*[rima92*]->services*[autofs*]]% set autostart yes
[rima92->device*[rima92*]->services*[autofs*]]% commit
[rima92->device[rima92]->services[autofs]]% category use default
[rima92->category[default]]% services
[rima92->category[default]->services]% add autofs
[rima92->category*[default*]->services*[autofs*]]% set autostart yes
[rima92->category*[default*]->services*[autofs*]]% commit
[rima92->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 235.

The rest of this section describes the configuration of NFS for static mounts, using RIMA View or `cmsh`.

Sections 3.10.1 and 3.10.2 explain how exporting and mounting of filesystems is done in general by an administrator using RIMA View and `cmsh`, and considers some mounting behavior that the administrator should be aware of.

Section 3.10.3 discusses how filesystems in general on a diskless node can be replaced via mounts of NFS exports.

Section 3.10.4 discusses how the root (/) filesystem on a diskless node can be replaced via mounts of NFS exports.

Section 3.10.5 discusses how OFED InfiniBand or iWarp drivers can be used to provide NFS over RDMA.

3.10.1 Exporting A Filesystem Using RIMA View And `cmsh`

Exporting A Filesystem Using RIMA View

As an example, if an NFS volume exists at "`rima92:/modeldata`" it can be exported using RIMA View using the head node clickpath:

`Devices`→`Head nodes[rima92]`→`Settings[JUMP TO]`→`Filesystem exports`

This shows the list of exports (figure 3.13):

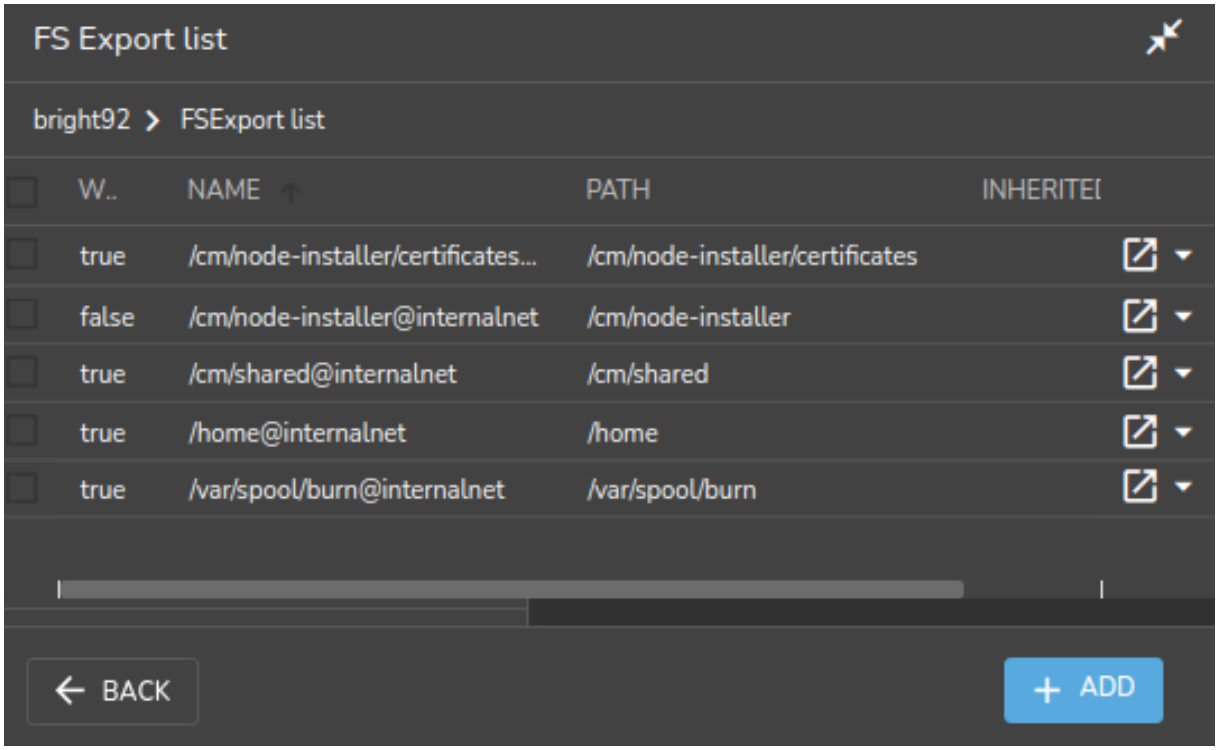


Figure 3.13: NFS exports from a head node viewed using RIMA View

Using the `Add` button, and selecting `FSExport` from the popup, a new entry (figure 3.14) can be configured with values as shown:

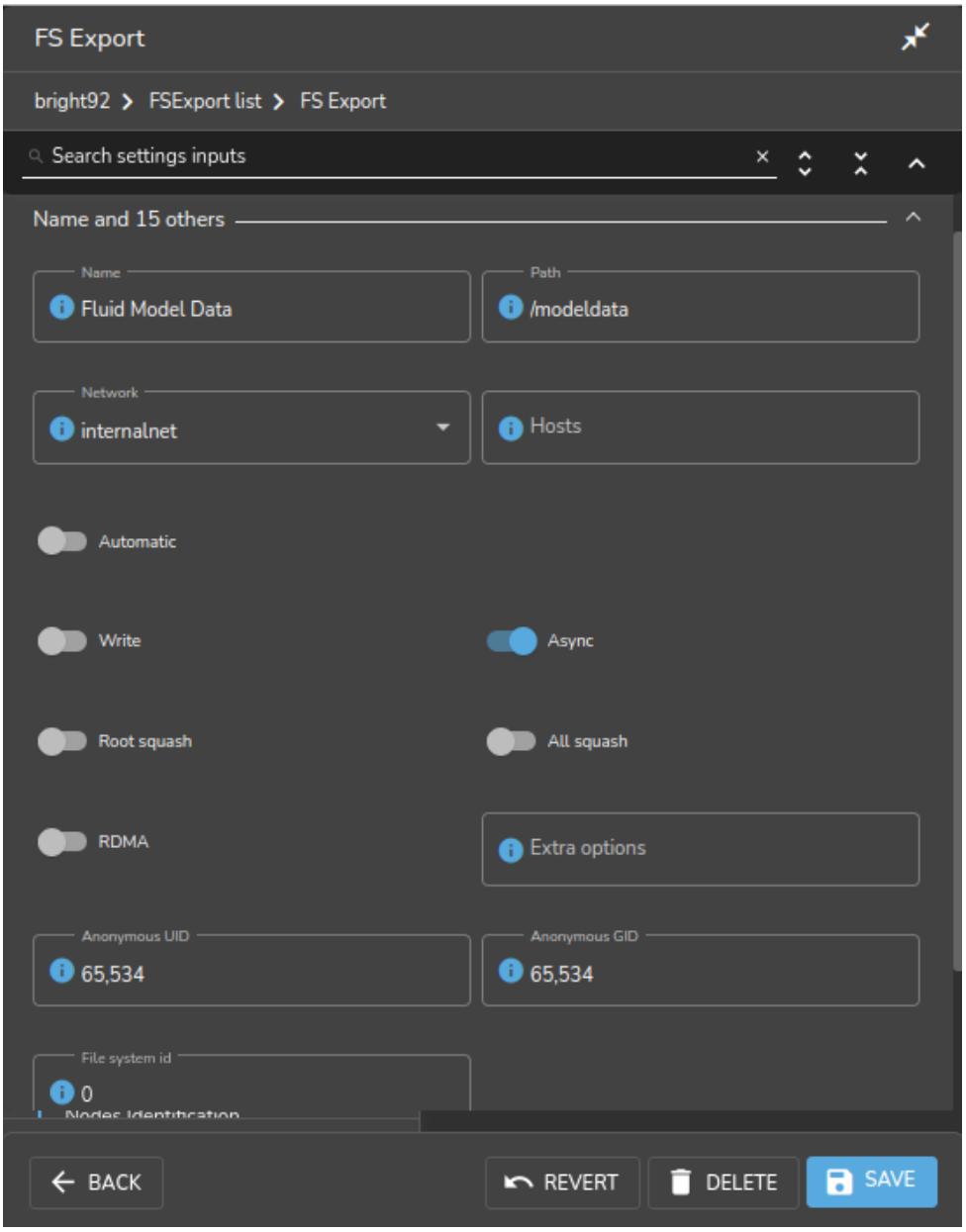


Figure 3.14: Setting up an NFS export using RIMA View

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 RIMA View NFS export procedure can be done in `cmsh` by using the `fsexports` submode on the head node (some output elided):

Example

```
[root@rima92 ~]# cmsh
[rima92]% device use rima92
[rima92->device[rima92]]% 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 RIMA Cluster 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 RIMA Cluster 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@rima92 ~]# cmsh
[rima92]% device use rima92
[rima92->device[rima92]]% roles
[rima92->device[rima92]->roles]% use headnode
[rima92->device[rima92]->roles[headnode]]% show
Parameter                        Value
-------------------------------- ----------------------
Name                             headnode
Revision
Type                             HeadNodeRole
Add services                     yes
Disable automatic exports        no
Provisioning associations        <2 internally used>
role use headnode
[rima92->device[rima92]->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 RIMA Cluster 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

```
[rima92->device[rima92]->fsexports]% list
Name (key)       Path                Network     Disabled
---------------- -------      ...    ----------- --------
opt              /opt                internalnet no
[rima92->device[rima92]->fsexports]% set opt disabled yes; commit
```

The reason for automating the export for nodes via a `headnode` role is that RIMA Cluster Manager 9.0 onward has multidistro and multiarch capabilities (section 7.7), which would make manual management of exports harder for such nodes. The reason for the extra hurdle of `disableautomaticexports` for built-ins is that that disabling these exports can result in an unbootable system.

3.10.2 Mounting A Filesystem Using RIMA 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 RIMA View

In RIMA View the clickpath to manage the mount points of a category such as `default` is:

`Grouping`→`Categories`[default]→`Edit`→`Settings[JUMP TO]`→`Filesystem mounts`

A mount point can be added with the `ADD` button, and clicking on the popup `FSMount`. Values for the remote mount point (`rima92:/modeldata`), the filesystem type (`nfs`), and the local mount point (`/modeldatagpu`) can then be set in category mode, while the remaining options stay at their default values (figure 3.15):

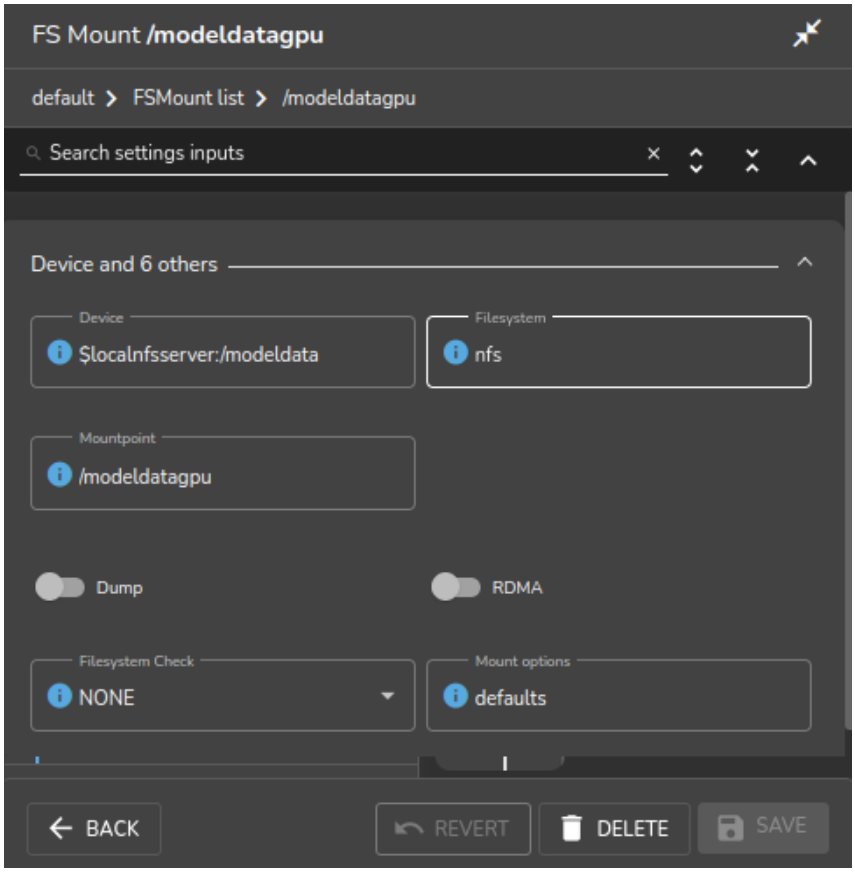


Figure 3.15: Setting up NFS mounts on a node category using RIMA View

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 RIMA 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@rima92 ~]# cmsh
[rima92]% category use default
[rima92->category[default]]% fsmounts
[rima92->category[default]->fsmounts]% add /modeldatagpu
[rima92->...*[/modeldatagpu*]]% set device rima92:/modeldata
[rima92->...*[/modeldatagpu*]]% set filesystem nfs
[rima92->category*[default*]->fsmounts*[/modeldatagpu*]]% commit
[rima92->category[default]->fsmounts[/modeldatagpu]]%
Device                  Mountpoint (key)     Filesystem
----------------------  -------------------  ----------
...
rima92:/modeldata     /modeldatagpu        nfs
[rima92->category[default]->fsmounts[/modeldatagpu]]% show
Parameter             Value
--------------------  ---------------------
Device                rima92:/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:

```
[rima92->...[/modeldatagpu]]% set mountoptions defaults,noac; commit
```

Mounting a CIFS might use:

```
[rima92->...[/modeldatagpu]]% set mountoptions gid,users,file_mode=0666,dir_mode=0777,\
iocharset=iso8859-15,credentials=/path/to/credential
[rima92->...[/modeldatagpu*]]% commit
```

A _netdev mount option to make systemd wait until the network is up before it is mounted can be added as follows:

```
[rima92->...[/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 3.10.1, the cluster manager 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 directive (Appendix C), 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 9.1), 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 RIMA View, for a node, inheritance by category is indicated in the clickpath 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@rima92 ~]# 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] rima92:/home /home                nfs
rima92:/cm/shared/exa+ /home/examples       nfs
[root@rima92 ~]#
```

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.

3.10.3 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 3.10.1 and 3.10.2. For the diskless case, the exported filesystem subtree is thus a particular path under /cm/images/<*image*>[2] on the provisioning node, and the subtree is mounted accordingly under / on the diskless node.

[2]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 RIMA Cluster Manager 9.2, the administrator should be aware that mounting certain paths such as `/bin` is not possible.

When RIMA 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@rima92 ~]# cmsh
[rima92]% device use rima92; fsexports
[rima92->device[rima92]->fsexports]% add defaultimage
[rima92...defaultimage*]]% set path /cm/images/default-image
[rima92...defaultimage*]]% set hosts 10.141.0.0/16
[rima92...defaultimage*]]% commit
[rima92...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
[rima92->category[default]]% fsmounts
[rima92->category[default]->fsmounts]% add /usr/share/doc
[rima92->...*[/usr/share/doc*]]% set device rima92:/cm/images/default-image/user/share/doc
[rima92->...*[/usr/share/doc*]]% set filesystem nfs
[rima92->category*[default*]->fsmounts*[/usr/share/doc*]]% commit
[rima92->category[default]->fsmounts[/usr/share/doc]]% list
Device                           Mountpoint (key)     Filesystem
---------------------            ------------------   ----------
...                              ...                  ...
rima92:/cm/images/usr/share/doc /usr/share/doc       nfs
[rima92->category[default]->fsmounts[/usr/share/doc]]% show
Parameter          Value
---------------    ------------------------------------------------
Device             rima92:/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 118 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@rima92 ~]# cd /cm/images/default-image/
[root@rima92 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@rima92 default-image]# cmsh
[rima92]% category use default; fsmounts
[rima92->category[default]->fsmounts]% list -f device:53,mountpoint:17
device                                              mountpoint (key)
--------------------------------------------------  -----------------
...                                                 ...
master:/cm/shared                                   /cm/shared
master:/home                                        /home
rima92:/cm/images/default-image/usr/share/locale    /usr/share/locale
rima92:/cm/images/default-image/usr/lib/jvm         /usr/lib/jvm
rima92:/cm/images/default-image/usr/share/doc       /usr/share/doc
rima92:/cm/images/default-image/usr/src             /usr/src
[rima92->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.

3.10.4 Mounting The Root Filesystem For A Diskless Node Over NFS

Mounting the root filesystem over NFS is a special case of mounting filesystem subtrees for a diskless node over NFS (section 3.10.3). The difference this time is that an initial root filesystem is deployed on the node via NFS as part of the standard Linux boot procedure. Some `tmpfs` mounts are then overlaid on top of parts of this filesystem.

The node being configured must have a disk setup that is diskless (section 3.9.4) for its node or node category.

If configuring a node category, it can be configured as follows on a CentOS 7 system:

1. The full diskless disk setup is partitioned first:

```
[root@rima92 ~]# cmsh
[rima92]% category use default
[rima92->category[default]]% set disksetup /cm/images/default-image/cm/local/apps/cmd/\
etc/htdocs/disk-setup/slave-diskless.xml
[rima92->category*[default*]]% commit   #puts full OS in RAM
```

2. Then the default image is exported to the internal network:

```
[rima92->category[default]]% device use master; fsexports
[rima92->device[rima92]->fsexports]% add /cm/images/default-image
...*]->fsexports*[/cm/images/default-image*]]% set hosts internalnet
...*]->fsexports*[/cm/images/default-image*]]% set name default@internal
...*]->fsexports*[default@internal*]]% commit   #exports default image
[rima92->device[rima92]->fsexports[default@internal]]% quit
```

3. The exported root filesystem is now mounted over the initial root filesystem:

```
[root@rima92 ~]# cmsh -c "category use default; fsmounts; add /; set\
 device master:/cm/images/default-image; set mountoptions defaults,ro;\
 set filesystem nfs; commit"        #adds readonly root via nfs
```

The root filesystem should be read-only during normal use. However, when the main system starts up after provisioning, a distribution start up script mounts the root filesystem as read-write. One way to get it back to a read-only state again is to set a finalize script (section 3.15.4) that arranges for the drive to remount itself as read-only after the system is fully up. The remount can be done with an `rc.local` file, which runs after all the other `rc` start up scripts have run. A suitable finalize script is then:

```
#!/bin/bash
echo "mount -o remount,ro /" >> /localdisk/etc/rc.local
```

4. The filesystem for / however has plenty of parts that need to be writable too. These are now mounted back in minimal subtrees into RAM. Keeping these mounts minimal means that the RAM used by the node is minimal. These read/write subtrees can be specified in a file, with one line per subtree:

```
[root@rima92 ~]# cat mountfiles      #for centos 7
/var/cache
/var/log
/var/tmp
/var/spool
/tmp
/dev
/cm/local/apps/cmd/etc
/cm/local/apps/openldap/etc
/cm/local/apps/sge/var
/cm/local/apps/slurm/var
/cm/local/apps/pbspro/var
/cm/local/apps/openpbs/var
/etc
/run
/var/lib #required by postfix and rpc-statd
```

The various workload manager `/var` directories for Grid Engine/Slurm/PBS variants are only needed if these workload managers are to be set up and used.

The subtrees can be mounted with a `for` loop:

```
[root@rima92 ~]# (echo "category use default; fsmounts"
  for i in $(<mountfiles)
  do
    echo "add $i; set device tmpfs; set filesystem tmpfs; exit"
  done
  echo "commit" ) | cmsh
```

If there is further software in the root filesystem that needs write access, that too should be mounted back into RAM.

The diskless nodes cannot be powered off with a simple `poweroff` or rebooted with a `reboot`. This is because the parts of the filesystem required to carry out these actions are unmounted before they are called in the diskless configuration. The `-f|--force` option to these commands forces a `poweroff` or `reboot`, but should only be used after first cleanly unmounting any writable shared filesystems, such as `/cm/shared` and `/home`. This is because the forcing options interrupt I/O syncing when invoked, which can result in a corrupted filesystem.

High Availability (Chapter 10) configured on the head nodes, with diskless nodes using a root filesystem mounted over NFS, would not work in a simple configuration such as this because it will result in stale NFS file handles over all the diskless nodes during a failover.

3.10.5 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 all the nodes, including the head node.

Drivers To Use For NFS over RDMA Must Be From The Parent Distribution

The use of the RDMA protocol (section 3.6) to provide NFS, by installing updated cluster manager OFED drivers (section 7.6 of the *Installation Manual*) is currently not supported. This is because these drivers are packaged by RIMA from the vendor (Mellanox or Qlogic) releases, and the vendor releases themselves do not support NFS over RDMA.

The option can be selected, but NFS will fall back to using the default NFS TCP/IP protocol.

When using NFS over RDMA, `ibnet`, the IP network used for InfiniBand, should be set. Section 3.6.3 explains how that can be done.

Exporting With RIMA 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 RIMA View is much the same as done in section 3.10.1 ("Exporting A Filesystem Using RIMA 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 3.10.1 ("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 3.10.1)

```
...
[...->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 RIMA View And `cmsh` Using NFS Over RDMA

The mounting of the exported filesystems using NFS over RDMA can then be done.

The procedure using RIMA View is largely like that in section 3.10.2, ("Mounting A Filesystem Using RIMA 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 3.10.2, ("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 3.10.2)

```
...
[rima92->category[default]->fsmounts]% add /modeldatagpu
[rima92->...*[/modeldatagpu*]]% set device rima92.ib.cluster:/modeldata
[rima92->...*[/modeldatagpu*]]% set filesystem nfs
[rima92->...*[/modeldatagpu*]]% set rdma yes
[rima92->category*[default*]->fsmounts*[/modeldatagpu*]]% commit
...
```

3.11 Managing And Configuring Services

3.11.1 Why Use The Cluster Manager For Services?

The unix services can be managed from the command line using the standard distribution tools:

```
service <service name> start|stop|status|...
```

and

```
systemctl list-unit-files <service name>.service
```

where <*service name*> indicates a service such as `mysqld`, `mariabd`, `nfs`, `postfix` and so on.

Already installed services can also be brought under cluster manager control, and started and stopped with RIMA 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 the cluster manager, then no attempts are made to restart the service. Attempted autostarts become possible again only after the cluster manager starts the service again.
  - If `autostart` is set to `on`, and if a service is removed using the cluster manager, 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 74.
- `runif`: (only honored for nodes used as part of a high availability configuration (chapter 10)) 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 10.5.3).

The details of a service configuration remain part of the configuration methods of the service software itself.

- Thus the cluster manager 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.

The cluster manager 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 10.4.6). The details of how to do this will depend on the service.

3.11.2 Managing And Configuring Services—Examples

If, for example, the CUPS software is installed ("`yum install cups`"), 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@rima92 ~]# chkconfig cups on
... symlinks created...
[root@rima92 ~]# service cups start
Redirecting to /bin/systemctl start cups.service
```

Managing The Service From `cmsh`

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 (some lines elided):

```
[rima92]% device services node001
[rima92->device[node001]->services]% add cups
[rima92->device*[node001*]->services*[cups*]]% show
Parameter                      Value
------------------------------ ------------------------
Autostart                      no
Belongs to role                no
Monitored                      no
...
Run if                         ALWAYS
Service                        cups
...
[rima92->device*[node001*]->services*[cups*]]% set monitored on
[rima92->device*[node001*]->services*[cups*]]% set autostart on
[rima92->device*[node001*]->services*[cups*]]% commit
[rima92->device[node001]->services[cups]]%
Apr 14 14:02:16 2017 [notice] node001: Service cups was started
[rima92->device[node001]->services[cups]]%
```

Other service options in `cmsh`: Within `cmsh`, the `start`, `stop`, `restart`, and `reload` options to the

`service <`*service name*`> start|stop|restart|...`

command can be used to manage the service at the `services` submode level. For example, continuing with the preceding session, stopping the CUPS service can be done by running the `cups` service command with the `stop` option as follows:

```
[rima92->device[node001]->services[cups]]% stop
Fri Apr 14 14:03:40 2017 [notice] node001: Service cups was stopped
Successfully stopped service cups on: node001
[rima92->device[node001]->services[cups]]%
```

The service is then in a `STOPPED` state according to the `status` command.

```
[rima92->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 RIMA View" on page 128.

Continuing from the preceding session, the CUPS service can also be added for a node category from category mode:

```
[rima92->device[node001]->services[cups]]% category
[rima92->category]% services default
[rima92->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

```
[rima92->category*[default*]->services*[cups*]]% show
...
[rima92->category*[default*]->services*[cups*]]% set autostart yes
[rima92->category*[default*]->services*[cups*]]% set monitored yes
[rima92->category*[default*]->services*[cups*]]% commit
[rima92->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
[rima92->category[default]->services[cups]]% status
node001..... cups            [STOPPED ]
node002..... cups            [   UP   ]
node003..... cups            [   UP   ]
node004..... cups            [   UP   ]
node005..... cups            [   UP   ]
```

Managing The Service From RIMA View

Using RIMA View, a service can be managed from an OSServiceConfig list window, accessible via the Services button from the JUMP TO section of the Settings. The window is accessible for

- Head Nodes, for example via a clickpath of
  Devices→Head Nodes[rima92]→Settings→Services
- Nodes, for example via a clickpath of
  Devices→Nodes[node001]→Settings→Services
  (figure 3.16):

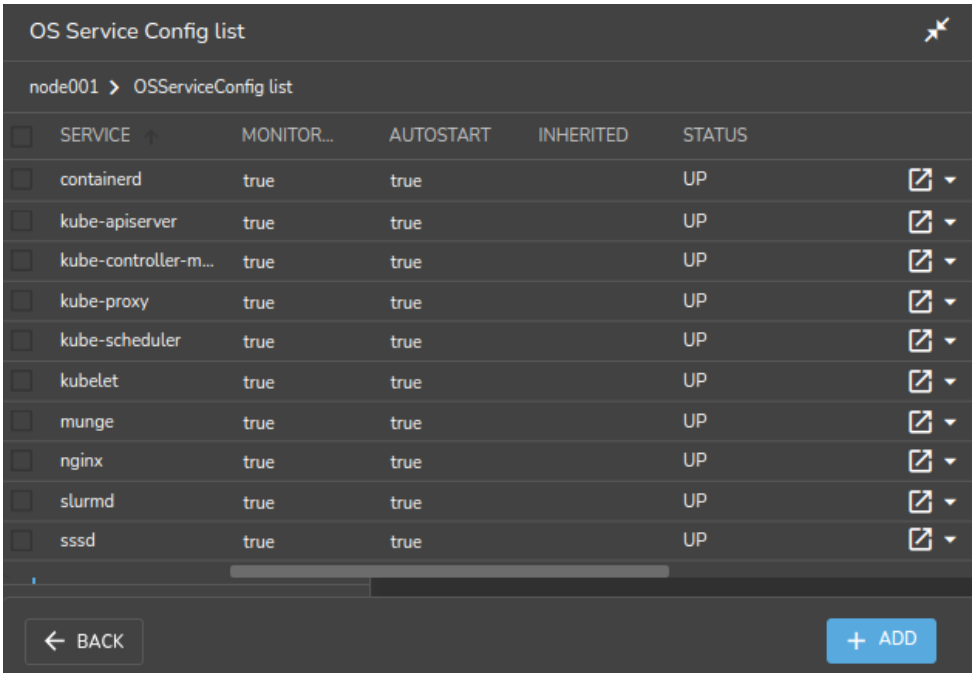


Figure 3.16: Operating system service configuration list window for nodes in RIMA View

- `Node categories`, for example via a clickpath of
  `Grouping`→`Node categories[default]`→`Settings`→`Services`

By default, with the `default` software image, there are no services set at category level for nodes (figure 3.17):

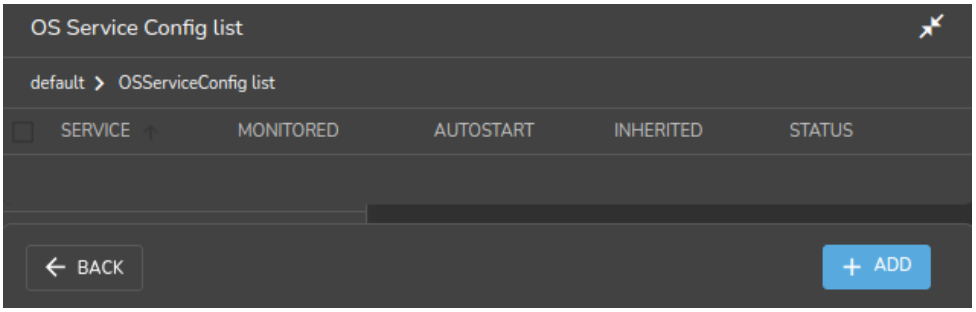


Figure 3.17: Operating system service configuration list window for the `default` category in RIMA View

The `Service` <*service name*> `start|stop|restart...` command options `start`, `stop`, `restart`, and so on, are displayed as selection options to an `OSService` popup that appears when the `ACTIONS` button is clicked (figure 3.18):

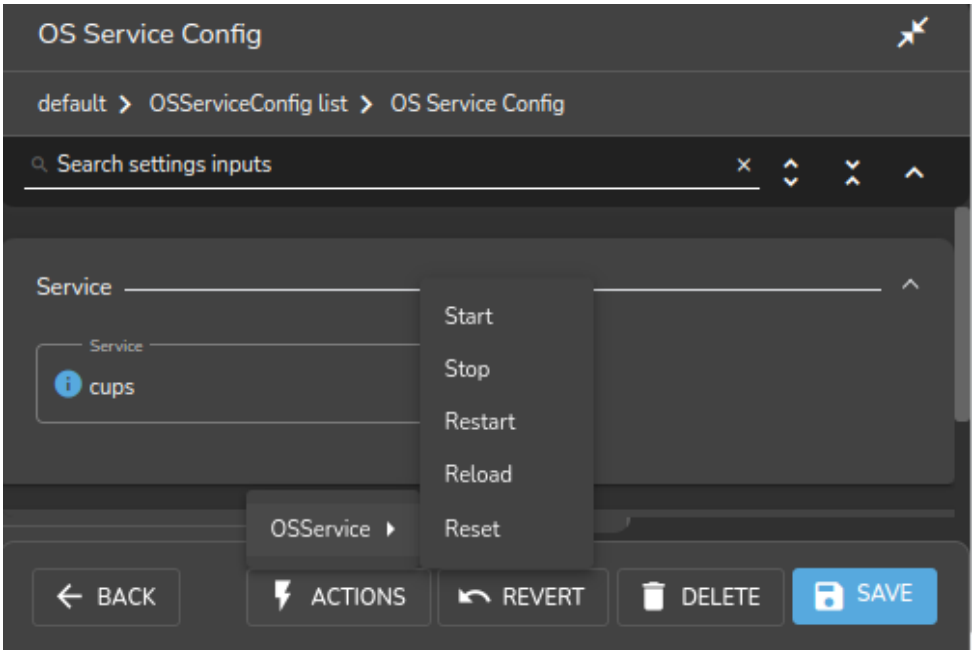


Figure 3.18: Operating system service actions in RIMA View

A service can be added with the `ADD` button, and clicking on the `OSServiceConfig` popup. The fields of the service can then be edited. The `REVERT` button reverts unsaved changes, while the `DELETE` button removes the saved changes.

Figure 3.19 shows CUPS being set up from an `Add` dialog in the services window. The window is accessible via the `ADD` button of figure 3.17.

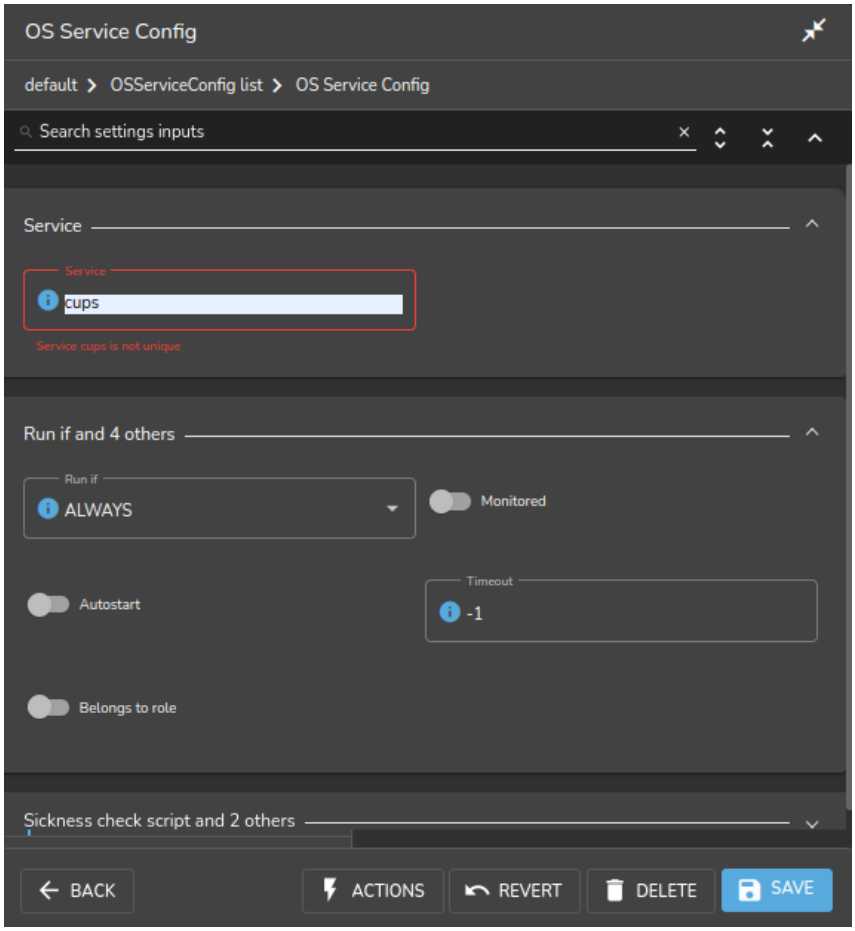


Figure 3.19: Setting up a service using RIMA View

For a service in the services subwindow, clicking on the `Status` button in figure 3.17 displays a grid of the state of services on a running node as either `Up` or `Down`.

Monitoring A Service With `cmsh` And RIMA 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 RIMA 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 F.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 8.6.3) 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 the cluster manager 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 the cluster manager 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.

Removing A Service From CMDaemon Control Without Shutting It Down

Removing a service from cluster manager control while `autostart` is set to `on` stops the service on the nodes:

```
[rima92->category[default]->services]% add cups
[rima92->category*[default*]->services*[cups*]]% set monitored on
[rima92->category*[default*]->services*[cups*]]% set autostart on
[rima92->category*[default*]->services*[cups*]]% commit; exit
[rima92->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.

```
[rima92->category[default]->services]% add cups
[rima92->category*[default*]->services*[cups*]]% set monitored on
[rima92->category*[default*]->services*[cups*]]% set autostart off
[rima92->category*[default*]->services*[cups*]]% commit; exit
Wed May 23 12:54:40 2012 [notice] node001: Service cups was started
[rima92->category[default]->services]% remove cups; commit
[rima92->category[default]->services]% !# no change: cups stays up
```

3.12 Managing And Configuring A Rack

3.12.1 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 RIMA View: The `Rack list` pane can be opened up in RIMA View via the clickpath `Datacenter Infrastructure`→`Racks` (figure 3.20):



Figure 3.20: Rack list using RIMA 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, and then clicking on the `Rack` popup. This opens a `Settings` tab in the rack item window pane where rack configuration can be carried out and saved (figure 3.21).

- an existing rack item can be edited with the `Edit` menu option, or by double-clicking on the item itself. This also opens up the `Settings` tab in the rack item window pane where rack configuration can be managed.

Racks overview in `cmsh`: The `rack` mode in `cmsh` allows racks defined in the cluster manager to be listed:

```
[rima92->rack]% list
Name (key)      Room           x-Coordinate   y-Coordinate   Height
--------------  -------------  -------------  -------------  ------
rack2           skonk works    2              0              42
racknroll                      1              0              42
```

Rack Configuration Settings

Rack configuration settings in RIMA View: A `Settings` tab for editing a rack item selected from the `Rack list` pane is shown in figure 3.21.

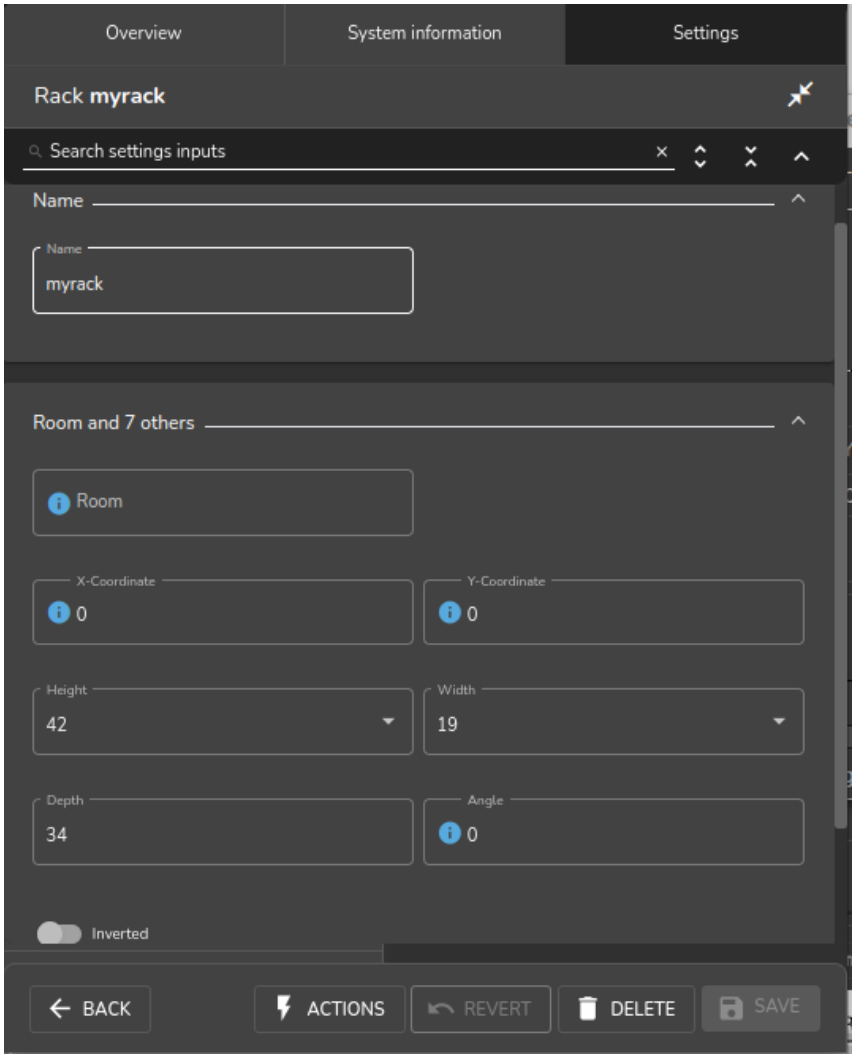


Figure 3.21: Rack configuration settings using RIMA View

The rack item configuration settings 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: In cmsh, tab-completion suggestions for the set command in rack mode display the racks available for configuration. On selecting a particular rack (for example, rack2 as in the following example), tab-completion suggestions then display the configuration settings available for that rack:

Example

```
[rima92->rack]% set rack
rack1  rack2  rack3
[rima92->rack]% set rack2
height        name          revision      width         y-coordinate
inverted      notes         room          x-coordinate
```

The configuration settings for a particular rack obviously match with the parameters associated with and discussed in figure 3.21.

Setting the values can be done as in this example:

Example

```
[rima92->rack]% use rack2
[rima92->rack[rack2]]% set room "skonk works"
[rima92->rack*[rack2*]]% set x-coordinate 2
[rima92->rack*[rack2*]]% set y-coordinate 0
[rima92->rack*[rack2*]]% set inverted no
[rima92->rack*[rack2*]]% commit
[rima92->rack[rack2]]%
```

3.12.2 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 RIMA View

Using RIMA View, a device such as a node node001 can be assigned to a rack via the clickpath Devices→Nodes[node001]→Settings→JUMP TO→Rack (figure 3.22):

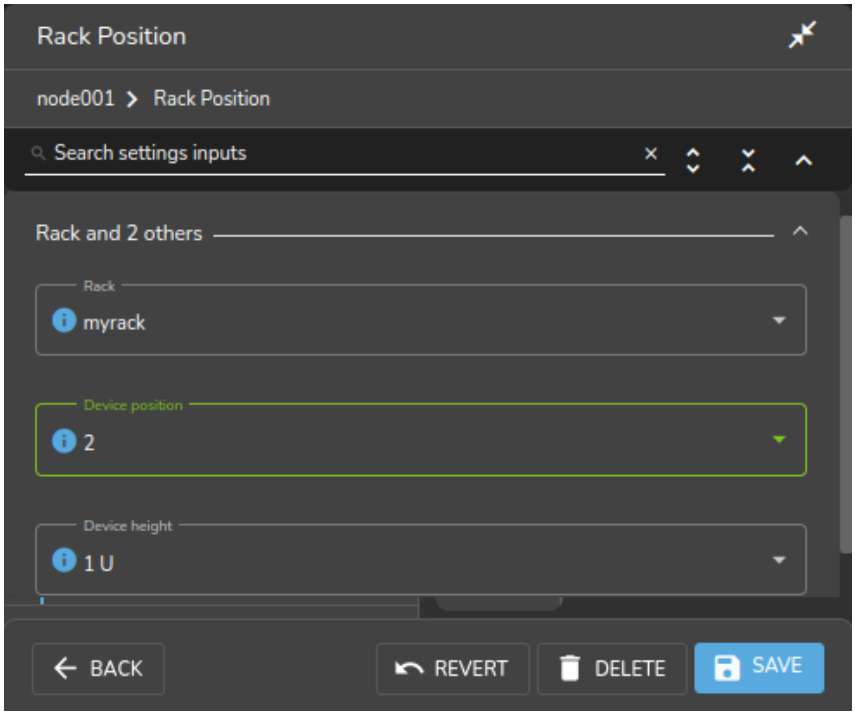


Figure 3.22: Rack assignment using RIMA View

Assigning Devices To A Rack Using `cmsh`

Using `cmsh` in `device` mode, node assignment to an existing rack `rack2`, of:

- `node001` to position 1
- `node002` to position 2
- `node003` to position 3

can conveniently be done as follows:

```
root@rima92:~# for i in 1..3 ; do cmsh -c "device use node00$i; get hostname; set rack rack2; \
get rack; set deviceposition $i; get deviceposition; commit"; done
```

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 RIMA 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 3.12.3), 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.

3.12.3 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 list in RIMA View: The `Chassis list` pane can be opened up in RIMA View via the click-path

`Datacenter Infrastructure`→`Chassis`

or

`Devices`→`Chassis`

A chassis can then be added, removed, or edited from the pane.
Within the `Chassis list` pane:

- a new chassis item can be added with the `ADD` button, and then clicking on the `Chassis` popup. This opens a `Chassis` pane where chassis configuration can be carried out and saved (figure 3.23).
- an existing chassis item can be edited with the `Edit` menu option, or by double-clicking on the item itself. This also opens up the `Chassis` pane for chassis configuration.

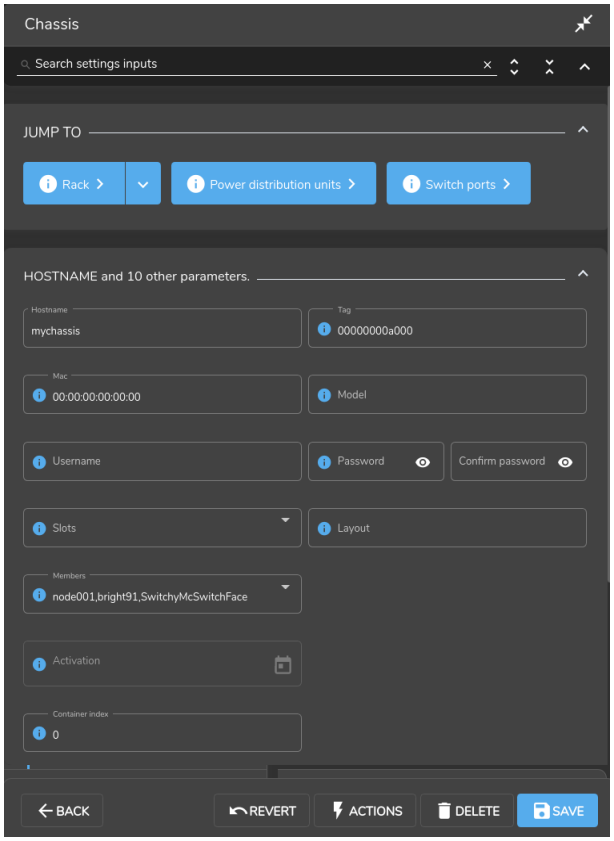


Figure 3.23: RIMA View chassis configuration

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
- `Tag`: a hardware tag for the chassis

- `Mac`: the MAC address of the chassis
- `Model`: the hardware model name
- `Rack`: the rack in which the chassis is placed
- `Members`: the `Members` menu option allows devices to be assigned to a chassis (figure 3.24). An item within the `Device` set can be any item from the subsets of `Node`, `Switch`, `Power Distribution Unit`, `Generic Device`, `Rack Sensor`, and `Gpu Unit`. These items can be filtered for viewing, depending on whether they are `Assigned` (members of the chassis), `Not Assigned` (not members of the chassis), or they can `All` be viewed (both `Assigned` and `Not Assigned` items).
- `Layout`: how the nodes in a chassis are laid out visually.
- `Network`: which network the chassis is attached to.
- `Username`, `Password`: the user name and password to access the chassis operating system
- `Power control`, `Custom power script`, `Custom power script argument`: power-related items for the chassis.
- `Userdefined1`, `Userdefined2`: administrator-defined variables that can be used by CMDaemon.

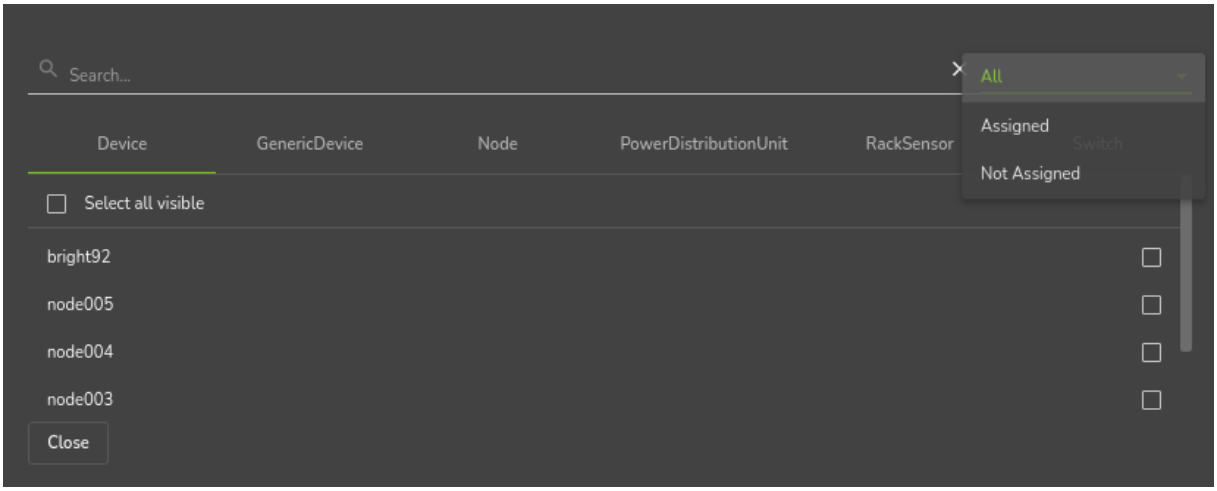


Figure 3.24: RIMA View Chassis Members Menu Options

Basic chassis configuration and node assignment with `cmsh`: The `chassis` mode 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

```
[rima92->device[chassis1]]% set
containerindex              hostname                  partition                 switchports
custompingscript            ip                        password                  tag
custompingscriptargument    layout                    powercontrol              userdefined1
custompowerscript           mac                       powerdistributionunits    userdefined2
custompowerscriptargument   members                   rack                      userdefinedresources
defaultgateway              model                     revision                  username
deviceheight                network                   slots
deviceposition              notes                     supportsgnss
```

Whether the suggested parameters are actually supported depends on the chassis hardware. For example, if the chassis has no network interface of its own, then the `ip` and `mac` address settings may be set, but cannot function.

The positioning parameters of the chassis within the rack can be set as follows with `cmsh`:

Example

```
[rima92->device[chassis1]]% set rack rack2
[rima92->device*[chassis1*]]% set deviceposition 1; set deviceheight 6
[rima92->device*[chassis1*]]% commit
```

The members of the chassis can be set as follows with `cmsh`:

Example

```
[rima92->device[chassis1]]% append members rima92 node001..node005
[rima92->device*[chassis1*]]% commit
```

3.13 Configuring GPU Settings

3.13.1 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.

3.13.2 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@rima92 ~]# cmsh
[rima92]% device use node001
[rima92->device[node001]]% gpusettings
[rima92->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@rima92 ~]# cmsh
[rima92]% category use default
[rima92->category[default]]% gpusettings
[rima92->category[default]->gpusettings]% list
GPU range (key)  Power limit  ECC mode     Compute mode  Clock speeds
---------------- ------------ ------------ ------------- -------------
[rima92->category[default]->gpusettings]% add nvidia 1-3 ; commit
[rima92->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 7.4.1 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 the cluster manager. 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 higher. 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 591), and corrected automatically.
  - Double bit errors are also detected, using the `EccDBitGPU` metric (page 591), 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.

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 7.5 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

```
[rima92->device[node001]->gpusettings]% list
Type GPU range Info
---- --------- ------------------
AMD  0         PowerPlay: manual
[rima92->device[node001]->gpusettings]% use amd:0
[rima92->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

```
[rima92->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 RIMA View

In RIMA 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 RIMA View: A `GPU Settings list` pane can be opened up in RIMA View for a regular node, for example `node001`, with the clickpath:
`Devices→Nodes[node001]→Edit→Settings→JUMP TO→GPU Settings`

Similarly, a `GPU Settings list` pane can be opened up in RIMA View for nodes in a category, for example `gpunodes`, with the clickpath:
`Grouping→Categories[gpunodes]→Edit→Settings→JUMP TO→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 `AMDGPUSettings` or the `NVIDIAGPUSettings` item in the popup. This opens an `AMDGPU Settings` pane or an `NVIDIA GPU Settings` pane, where GPU configuration can be carried out and saved (figure 3.25).

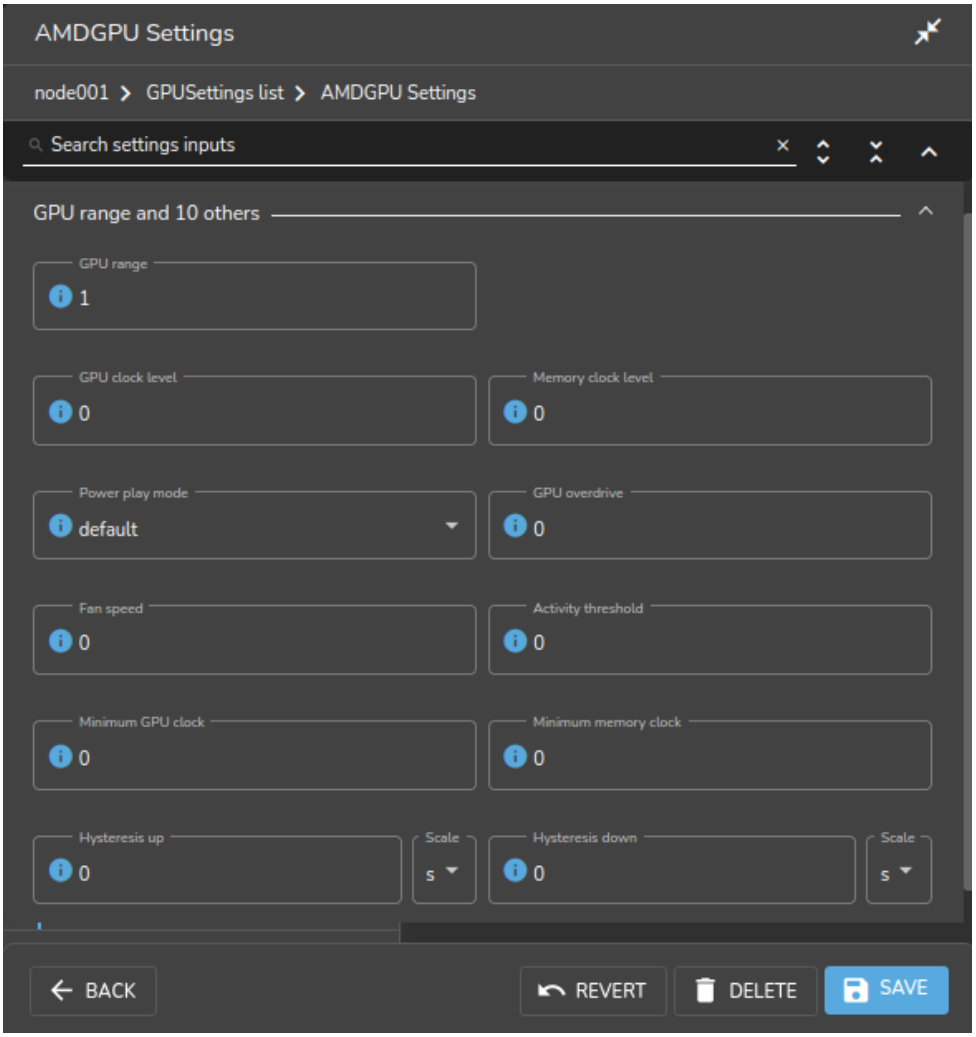


Figure 3.25: GPU settings window for a node

- an existing GPU settings item can be edited with the `Edit` menu option, or by double-clicking on the item itself. This also 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.

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@rima92 ~]$ 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

```
[rima92->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
[rima92->configurationoverlay]% use slurm-client
[rima92->configurationoverlay[slurm-client]]% roles
[rima92->configurationoverlay[slurm-client]->roles]% use slurmclient
[rima92->configurationoverlay[slurm-client]->roles[slurmclient]]% genericresources
[rima92->configurationoverlay[slurm-client]->roles[slurmclient]->genericresources]%
[rima92->...->roles[slurmclient]->genericresources]% add gpu0
[rima92->...->roles*[slurmclient*]->genericresources*[gpu0*]]% set name gpu
[rima92->...->roles*[slurmclient*]->genericresources*[gpu0*]]% set file /dev/nvidia0
[rima92->...->roles*[slurmclient*]->genericresources*[gpu0*]]% commit
[rima92->...->roles[slurmclient]->genericresources[gpu0]]%
    (Repeat similar settings for the other GPUs,  gpu1...gpu7)
[rima92->...->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
[rima92->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

```
[rima92->wlm[slurm]]% jobqueue
[rima92->wlm[slurm]->jobqueue]% use defq
[rima92->wlm[slurm]->jobqueue[defq]]% set oversubscribe yes:8
[rima92->wlm[slurm]->jobqueue*[defq*]]% commit
[rima92->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.

PBS: RIMA Cluster Manager version 9.0 onward supports GPU configuration in PBS via the `cm-wlm-setup` tool after installation (section **??**).

Grid Engine: GPU configuration is carried out as follows:

1. **Grid Engine cgroups parameter modifications for GPU devices (obsolete)**
   In RIMA Cluster Manager 8.0, 8.1, and 8.2: The command `qconf -mconf` <*hostname*> is used to add GPU device(s) to the `cgroups_params` parameter for the host.

   For example, if `node001` has access to 8 GPUs, then running the command `qconf -mconf node001` on the head node may initially show a configuration such as:

   ```
   hostname                node001
   cgroups_params          cgroup_path=/sys/fs/cgroup cpuset=true \
                           mount=false freezer=true freeze_pe_tasks=true \
                           killing=false forced_numa=false \
                           h_vmem_limit=false m_mem_free_hard=false \
                           m_mem_free_soft=false min_memory_limit=0
   ```

   The GPU devices can then be appended to the end of the last line and saved, so that the last line becomes:

   Example

   ```
                           m_mem_free_soft=false min_memory_limit=0 devices=/dev/nvidia[0-7]
   ```

   The parameter can be made persistent by setting the `AdvancedConfig` directive (page 528) to include the setting `UGEDisableCgroupSettings=1` and restarting CMDaemon.

2. **GPU devices configuration in `ugeclient` role (current)**
   In RIMA Cluster Manager 9.0: setting the `AdvancedConfig` directive (page 528) to include the setting `UGEDisableCgroupSettings=1` is not needed. This is because the `qconf -mconf` configuration method is not required.

   Instead, RIMA Cluster Manager 9.0 introduces specification of the GPU devices via the `devices` parameter for the `ugeclient` role:

   Example

```
[rima92->category[dgx]->roles[ugeclient]]% set gpus 8
[rima92->category*[dgx*]->roles*[ugeclient*]]% set gpudevices gpu0[device=/dev/nvidia0,cuda_id=0]
[rima92->category*[dgx*]->roles*[ugeclient*]]% append gpudevices gpu1[device=/dev/nvidia1,cuda_id=1]
[rima92->category*[dgx*]->roles*[ugeclient*]]% append gpudevices gpu2[device=/dev/nvidia2,cuda_id=2]
[rima92->category*[dgx*]->roles*[ugeclient*]]% append gpudevices gpu3[device=/dev/nvidia3,cuda_id=3]
[rima92->category*[dgx*]->roles*[ugeclient*]]% append gpudevices gpu4[device=/dev/nvidia4,cuda_id=4]
[rima92->category*[dgx*]->roles*[ugeclient*]]% append gpudevices gpu5[device=/dev/nvidia5,cuda_id=5]
[rima92->category*[dgx*]->roles*[ugeclient*]]% append gpudevices gpu6[device=/dev/nvidia6,cuda_id=6]
[rima92->category*[dgx*]->roles*[ugeclient*]]% append gpudevices gpu7[device=/dev/nvidia7,cuda_id=7]
[rima92->category*[dgx*]->roles*[ugeclient*]]% commit
[rima92->category[dgx]->roles[ugeclient]]%.
```

3. **GE port configuration for DCGM**
   The DCGM port should be set correctly.

   In RIMA Cluster Manager versions prior to 9.0, it can be changed from its default with the command `qconf -mconf`. The format for the setting is:

   Example

```
execd_params UGE_DCGM_PORT=<port>
```

   The default value for <*port*> is `5555`.

   From RIMA Cluster Manager 9.0 onward, setting the DCGM port for GE is a configuration option in the CMDaemon front ends.

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

```
[rima92->wlm[lsf]]% set gpuautoconfig yes
[rima92->wlm*[lsf*]]%
```

The parameter can also be set during LSF configuration via `cm-wlm-setup` (figure 3.26):

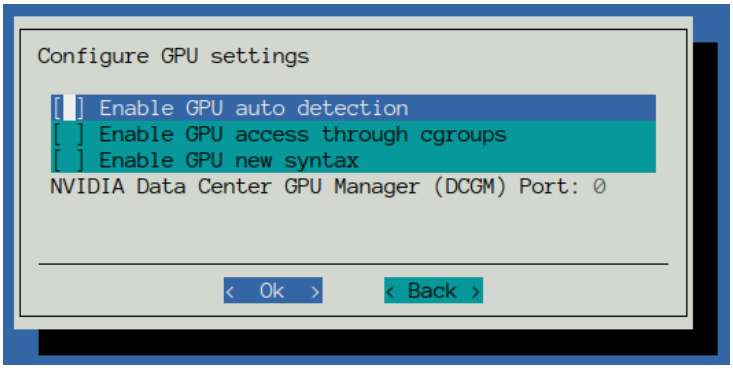


Figure 3.26: GPU settings screen for LSF in cm-wlm-setup

GPU resource enforcement can be configured for LSF as follows:

Example

```
[rima92->wlm[lsf]->cgroups]% append resourceenforce gpu
[rima92->wlm*[lsf*]->cgroups*]% commit
[rima92->wlm[lsf]->cgroups]%
```

3.13.3 MIG Configuration

MIG configuration can be carried out for the cluster using the RIMA MIG management as described in this section.

An alternative for MIG configuration is to not use the RIMA 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-RIMA MIG management tools are used, then the cluster manager 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@rima92:~# 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 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

```
rima92->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
...
rima92->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
...

[rima92->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 $1/8^{th}$ 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 $1/7^{th}$ 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

```
[rima92->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

```
[rima92->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):

```
[rima92->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

```
[rima92->device[node001]]% reboot
Reboot in progress for: node001
...
node001 [   UP   ]
[rima92->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, 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

```
[rima92->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

```
[rima92->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 3.13.2):

Example

```
[rima92->device[node001]]% gpusettings
[rima92->device*[node001]->gpusettings]% list
Type   GPU range Info
------ --------- --------
Nvidia 0         default
[rima92->device[node001]->gpusettings]% use nvidia:0
[rima92->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

  ```
  [rima92->device[node001]->gpusettings[nvidia:0]]% set migprofiles 1g.5gb; commit
  [rima92->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

  ```
  [rima92->device[node001]->gpusettings[nvidia:0]]% ..;..
  [rima92->device[node001]]% mig show
  Node             GPU      MIG      Instance Name     Profile  Start    Size
  ---------------- -------- -------- -------- -------- -------- -------- --------
  ```

  Applying the profile deploys the configuration, and shows the configuration:

Example

```
[rima92->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

```
[rima92->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
[rima92->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

```
[rima92->device[node001]]% gpusettings; use nvidia:0
[rima92->device[node001]->gpusettings[nvidia:0]]% set migprofiles 7*1g.5gb; commit
[rima92->device[node001]->gpusettings[nvidia:0]]% ..;..
[rima92->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

```
[rima92->device[node001]]% gpusettings; use nvidia:0
[rima92->device[node001]->gpusettings[nvidia:0]]% set migprofiles 7g.40gb; commit
[rima92->device[node001]->gpusettings[nvidia:0]]% ..;..
[rima92->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

```
[rima92->device[node001]]% gpusettings; use nvidia:0
[rima92->device[node001]->gpusettings[nvidia:0]]% set migprofiles 2g.10gb; commit
[rima92->device[node001]->gpusettings[nvidia:0]]% ..;..
[rima92->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

```
[rima92->device[node001]]% gpusettings; use nvidia:0
[rima92->device[node001]->gpusettings[nvidia:0]]% set migprofiles 2g.10gb:0; commit
[rima92->device[node001]->gpusettings[nvidia:0]]% ..;..
[rima92->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

```
[rima92->device[node001]]% gpusettings; use nvidia:0
[rima92->device[node001]->gpusettings[nvidia:0]]% set migprofiles 2g.10gb:0:0; commit
[rima92->device[node001]->gpusettings[nvidia:0]]% ..;..
[rima92->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 documentation. For example, for the A30 profiles at:

`https://docs.nvidia.com/datacenter/tesla/mig-user-guide/#a30-profiles`

and for the A100 profiles at:

`https://docs.nvidia.com/datacenter/tesla/mig-user-guide/#a100-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

```
[rima92->device[node001]]% gpusettings; use nvidia:0
[rima92->device[node001]->gpusettings[nvidia:0]]% set migprofiles 3*2g.10gb:0:0; commit
[rima92->device[node001]->gpusettings[nvidia:0]]% ..;..
[rima92->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:

```
[rima92->device[node001]]% gpusettings; use nvidia:0
[rima92->device[node001]->gpusettings[nvidia:0]]% set migprofiles 2*1g.5gb 2g.10gb; commit
[rima92->device[node001]->gpusettings[nvidia:0]]% show
Parameter                      Value
------------------------------ ------------------------------------------------
...
MIG profiles                   2*1g.5gb,2g.10gb
[rima92->device[node001]->gpusettings[nvidia:0]]% ..;..
[rima92->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.

3.14 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 3.14.2)
- `customremoteconsolescript` and `customremoteconsolescriptargument` (section 3.14.3)

In addition to the preceding `custom*` scripts, system information scripts can be set that provide extra information to the `sysinfo` command in the cluster manager (section 3.14.4).

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.

3.14.1 `custompowerscript`

The use of custom power scripts is described in section 4.1.4.

3.14.2 `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@rima92]# cmsh -c "device; format hostname:15, status:15; list"
hostname (key)  status
--------------- ---------------
rima92        [   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.

3.14.3 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@rima92]# cmsh
[rima92]% device use node001
[rima92->device[node001]]% get customremoteconsolescript; get customremoteconsolescriptargument

[rima92->device[node001]]% set customremoteconsolescript /root/kvmaccesshack
[rima92->device[node001]]% set customremoteconsolescriptargument 1
[rima92->device[node001]]% rconsole
```

KVM console access session using the 1 *argument option is displayed*

In RIMA View, the corresponding clickpaths 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 clickpath:

Devices→Nodes↓Edit←Connect→Remote console

3.14.4 sysinfo Custom Scripts

Standard sysinfo

The sysinfo command in the cluster manager is run from device mode in cmsh for a node. By default, sysinfo returns some basic hardware information for the node.

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. 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@rima92 ~]# 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@rima92 ~]# /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":"rima92",
"rima92 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@rima92 ~]# cp -r /cm/local/apps/cmd/scripts/sysinfo/head/outputscript.sh \
 /cm/images/default-image/cm/local/apps/cmd/scripts/sysinfo/local/
[root@rima92 ~]# cmsh
[rima92]% device
[rima92->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

```
[rima92->device]% sysinfo node001 --update
[rima92->device]% sysinfo node001
...

Age                          11s
CMDaemon running on          node001                                                    }
script running on            node001                                                    }
node001 UUID                 3b1c7973-07ef-419f-9324-4edf999690d5                       } local script
script path is               /cm/local/apps/cmd/scripts/sysinfo/local/outputscript.sh   }


CMDaemon running on          node001                                                    }
script running on            rima92                                                     }
rima92 UUID                6733d33a-2933-41ea-aa3c-b218e784c8b9                         } head script
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 the cluster manager 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.

3.15 Cluster Configuration Without Execution By CMDaemon

3.15.1 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 RIMA 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 RIMA 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 3.15.2. 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 3.15.3.

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 3.15.4.

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 3.10.

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 7.2–7.6. Third-party software management from outside the repositories, for software that is part of the cluster manager is covered in Chapter 7 of the *Installation Manual*.

   Third-party software that is not part of the cluster manager 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.

3.15.2 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 3.15.2), the following methods can be used:

- in RIMA View, via `Cluster`→`Settings`→`Cluster name`
- 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 3.15.2).

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 3.15.4).

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@rima92 ~]# cmsh
   [rima92]% softwareimage
   [rima92->softwareimage]% clone default-image imagetwo
   [rima92->softwareimage*[imagetwo*]]% commit
   ... [notice] rima92: Started to copy: /cm/images/default-image -> /cm/images/imagetwo
   [rima92->softwareimage*[imagetwo*]]%
   ... [notice] rima92: Copied: /cm/images/default-image -> /cm/images/imagetwo
   [rima92->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 7.2–7.6.

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:

   ```
   [rima92->softwareimage[imagetwo]]% category
   [rima92->category]% clone default categorytwo
   [rima92->category*[categorytwo*]]% set softwareimage imagetwo
   [rima92->category*[categorytwo*]]% commit
   [rima92->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):

     ```
     [rima92->category[categorytwo]]% device
     [rima92->device]% use node099
     [rima92->device[node099]]% set category categorytwo
     [rima92->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

     ```
     [rima92->device]% foreach -n node100..node200 (set category categorytwo)
     [rima92->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. From RIMA Cluster Manager 8.2 onward, there is 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 the cluster manager 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.

3.15.3 Making All Nodes Function Differently From Normal Cluster Behavior With FrozenFile

Configuration changes carried out by RIMA 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 RIMA 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 523) 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.

3.15.4 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 3.15.2. 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 the cluster manager. 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 RIMA View clickpath:

`Devices`→`Nodes[node001]`→`Edit`→`Settings`→`Interfaces[BOOTIF]`→`Edit`→`Speed`

3.15.5 Examples Of Configuring Nodes With Or Without CMDaemon

A node or node category can often have its software configured in CMDaemon via RIMA View or `cmsh`:

Example

Configuring a software for nodes using RIMA View or `cmsh`: If the software under consideration is CUPS, then a node or node category can manage it from RIMA View or `cmsh` as outlined in section 3.11.2.

A counterexample to this is:

Example

Configuring a software for nodes without using RIMA View or `cmsh`[3], using an image: Software images can be created with and without CUPS configured. 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 RIMA 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 the cluster manager, 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 RIMA View Or `cmsh`: Category Settings

When configuring nodes using RIMA 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 RIMA 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 3.15.2 to implement a functionality such as CUPS would load up CUPS as configured in an image, and would not handle

[3] except to link nodes to their appropriate image via the associated category

it via CMDaemon[3]. So, in section 3.15.2, 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.

3.16 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@rima92 ~]# cmsh
[rima92]% device use node001
[rima92->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@rima92 ~]# cd /var/spool/cmd/
[root@rima92 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@rima92 cmd]# cd config_file_versions/node001/cm/local/modulefiles/slurm/
[root@rima92 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@rima92 slurm]#
```

4

Power Management

Aspects of power management in RIMA Cluster 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 10)
- allowing cluster burn tests to be carried out (Chapter 8 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 the cluster manager 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 briefly covers monitoring power.

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. The device 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. A summary of the configuration of PDUs is as follows:

The PDU can be added via `cmsh` using `device` mode, and is set as an object with a `type` of `PowerDeviceUnit` and is given a name. The value for `Ports` is automatically read, and is the number of power ports available to the other devices in the cluster.

In Bright View the corresponding clickpath for a PDU named `mypdu` is:

```
Devices→PowerDistribution Unit list→mypdu→Add
```

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.

For example, for a node `node001` that is be powered by the PDU `mypdu`, in Bright View the configuration can be done using the clickpath:

```
Devices→Nodes→Physical Node list[node001]→Edit→Settings→JUMP TO→Power
Distribution Units→ADD→PDUPort
```

which opens up the `PDU Port` window (figure 4.1):

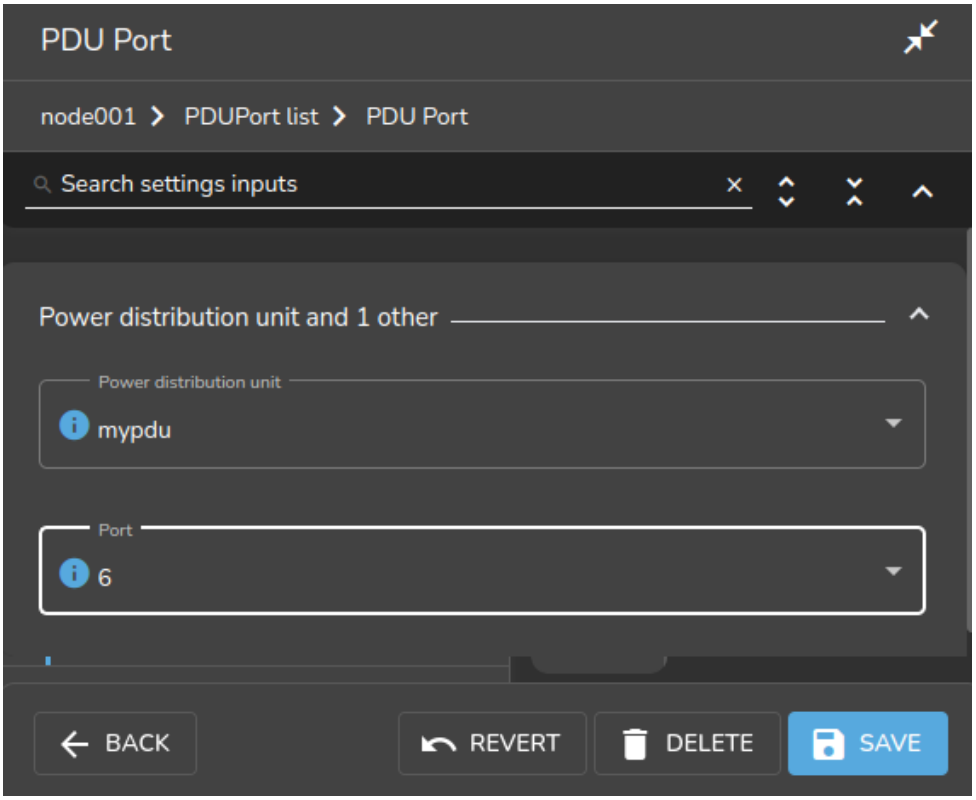


Figure 4.1: PDU configuration settings for a node

and allows the PDU and port used by `node001` to be set.

For the APC brand of PDUs: the `Power control` property (page 102) 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

```
[rima92->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 the cluster manager. It is suggested that administrators record that a dumb PDU is assigned to a node as follows:

- in RIMA View the Notes field or the Userdefined1/Userdefined2 fields can be used with the clickpaths:

  Devices*[device]*→Settings→Partition→Notes
  or
  Devices*[device]*→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

```
[rima92]% device use node001
[rima92->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

```
[rima92->device[node001]]% power status
```

  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:

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

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 102) 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 RIMA 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 RIMA 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". The cluster manager 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 100.

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 RIMA View or cmsh. There are four main power operations:

- Power On: power on a device
- Power Off: power off a device
- Power Reset: power off a device and power it on again after a brief delay
- Power Status: check power status of a device

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. 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 operations are:
  - `waiting`: waiting to be executed
  - `busy`: are being executed
  - `canceled`: have been canceled
  - `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 RIMA View

In RIMA View, executing the main power operations can be carried out as follows:

- via the menu dropdown for a node. For example:
  - for the head node, via the clickpath `Devices`→`Head Nodes`↓`Power`
  - for a regular node via the clickpath `Devices`→`Nodes`↓`Power`
- via the menu dropdown for a category or group. For example, for the `default` category, via the clickpath `Grouping`→`Categories`↓`Power`
- via the `Actions` button. The `Actions` button is available when specific device has been selected. For example, for the head node `rima92` the `Actions` button can be seen via the clickpath `Devices`→`Head Nodes[rima92]`→*checkbox*.

  Clicking on the `Actions` button then makes power operation buttons available (figure 4.2).

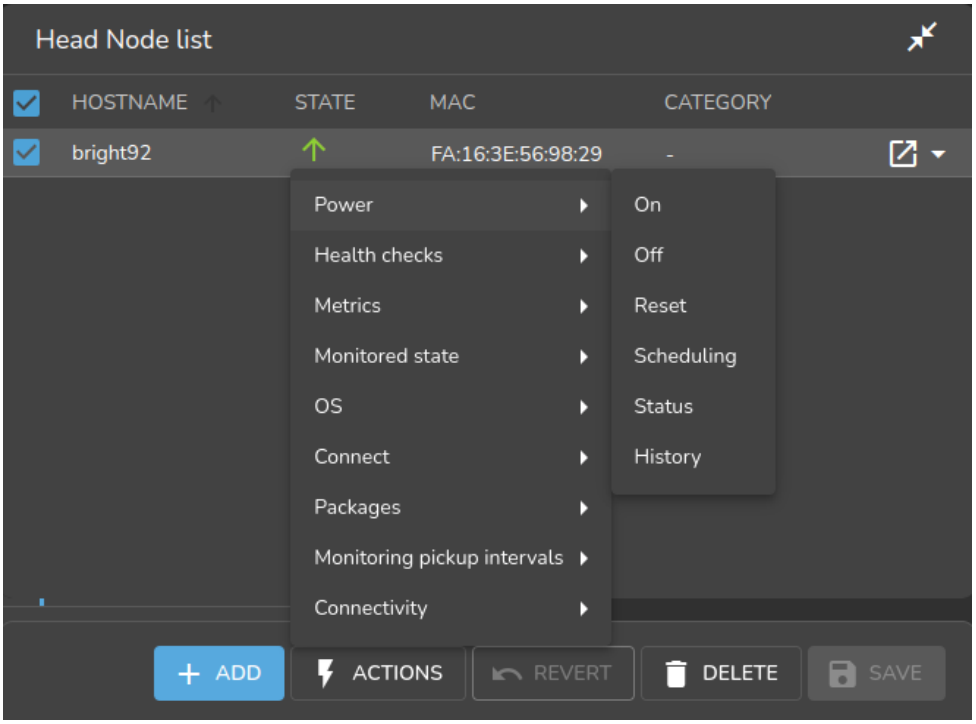


Figure 4.2: `Actions` button, accessing the power operations

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 45). 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 522).

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 is:

```
[rima92->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 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 the cluster manager 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 8 on cluster monitoring has more on metrics and how they can be visualized.

4.4 Switch Configuration To Survive 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 a node does not provision from 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 RIMA 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 the cluster manager. This is called a *network boot*. On the x86 architectures 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 188) 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

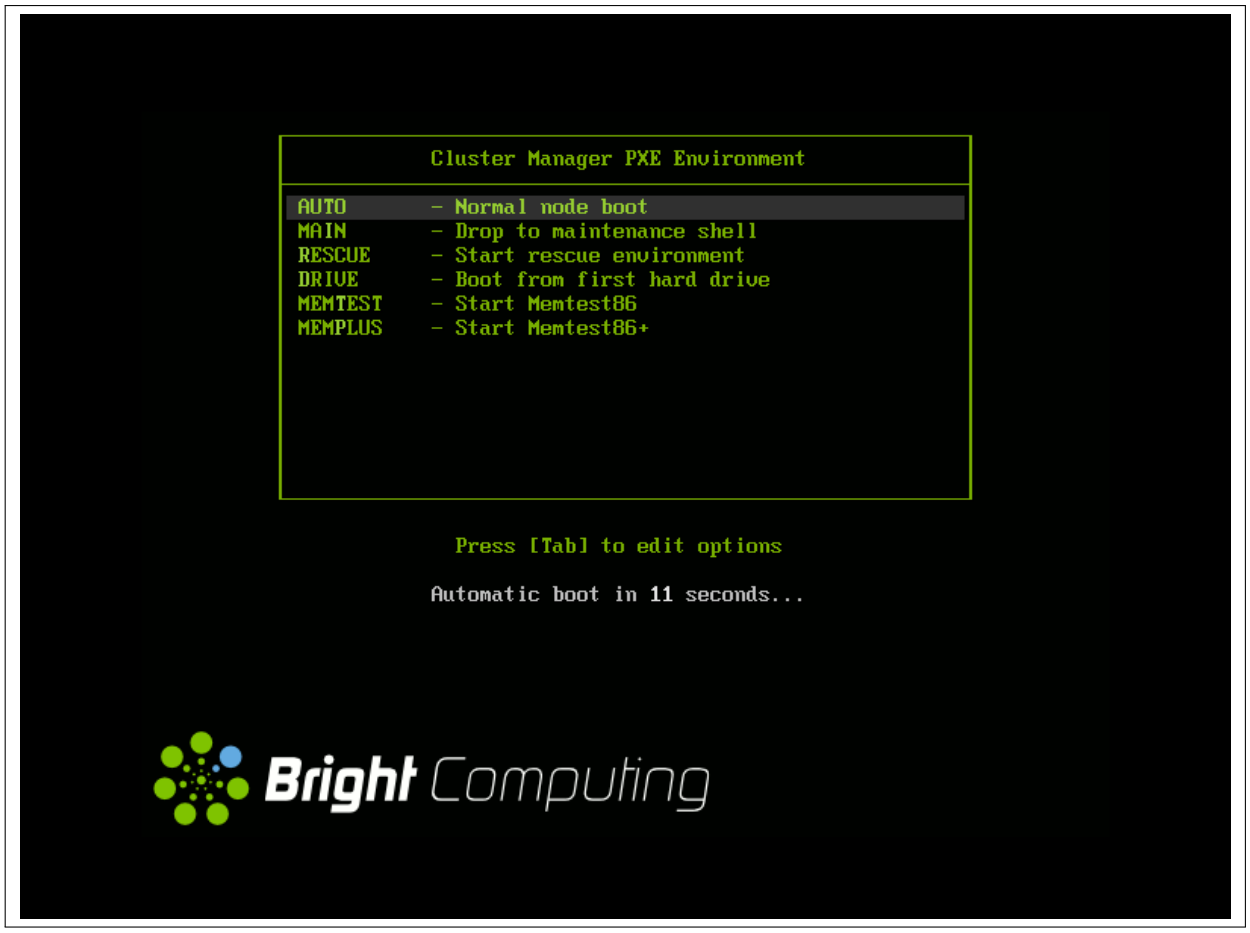


Figure 5.1: PXE boot menu options

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@rima92 ~]# cmsh
    [rima92]% device use node001
    [rima92->device[node001]]% set pxelabel MEMTEST ; commit
    ```

    Carrying it out for all nodes in the default category can be done, for example, with:

    Example

    ```
    [root@rima92 ~]# cmsh
    [rima92]% device
    [rima92->device]% foreach -c default (set pxelabel MEMTEST)
    [rima92->device*]% commit
    ```

    The value of pxelabel can be cleared with:

    Example

    ```
    [root@rima92 ~]# cmsh -c "device; foreach -c default (clear pxelabel); commit"
    ```

  - In RIMA View, the PXE label can be set for a node node001 using the clickpath

    ```
    Devices→Nodes→Physical Node list[node001]→Settings[Physical Node
    node001]→
    Provisioning[PXE Label]
    ```

    which leads to a screen as in (figure 5.2):

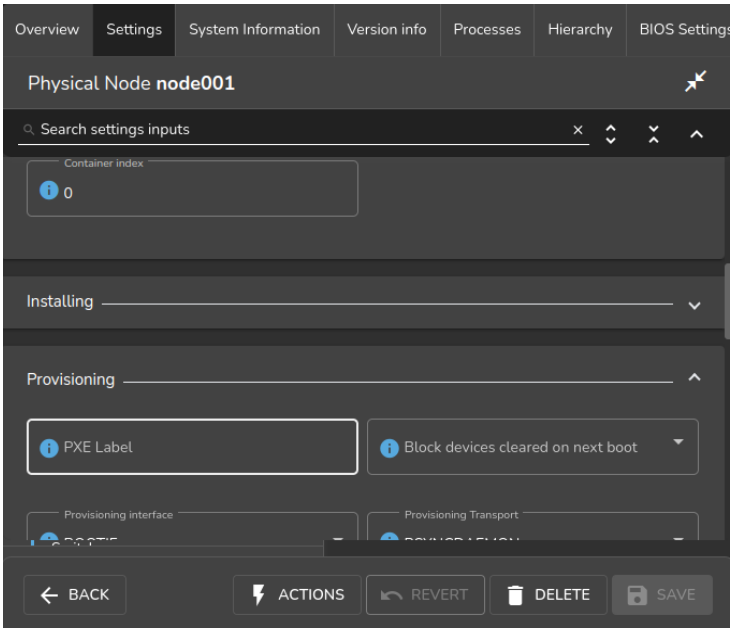


Figure 5.2: RIMA View `PXE Label` option

- 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 via RIMA 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 7.

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 via PXE is enabled by carrying out these 3 steps:

1. Making the cluster manager aware that nodes are to be booted over InfiniBand. Node booting (section 3.2.3, page 79) can be set from `cmsh` or RIMA 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 RIMA View: The `Settings` window for the InfiniBand network, for example `ibnet`, can be accessed from the `Networking` resource via the clickpath `Networking`→`Networks[ibnet]`→`Edit`→`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 3.6.3. 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. For PXE booting over Omni-Path, older Omni-Path cards may need to have UEFI firmware installed on them, and the only supported boot mode is then UEFI.

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 via 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 177).

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 via 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

```
[rima92->category[default-centos7-x86_64]]% get bootloaderprotocol
HTTP
```

It can be modified at category or node level, to one of the values `HTTP`, `HTTPS`, or `TFTP`:

Example

```
[rima92->device[node001]]% get bootloaderprotocol
HTTP (default-centos7-x86_64)
[rima92->device[node001]]% set bootloaderprotocol<TAB><TAB>
http https tftp
[rima92->device[node001]]% set bootloaderprotocol tftp
[rima92->device*[node001*]]% commit
[rima92->device[node001]]% get bootloaderprotocol
TFTP
```

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 via 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.

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

```
[rima92]% category add misc
[rima92->category*[misc*]]% roles
[rima92->category*[misc*]->roles]% assign provisioning
[rima92...*]->roles*[provisioning*]]% set allimages no
[rima92...*]->roles*[provisioning*]]% set localimages default-image
[rima92...*]->roles*[provisioning*]]% set provisioningslots 20
[rima92...*]->roles*[provisioning*]]% show
Parameter                        Value
-------------------------------- --------------------------------
All Images                       no
Include revisions of local images yes
Local images                     default-image
Name                             provisioning
Nodegroups
Provisioning associations        <0 internally used>
Revision
Shared images
Type                             ProvisioningRole
Provisioning slots               20
[rima92->category*[misc*]->roles*[provisioning*]]% commit
[rima92->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

```
[rima92]% device use node001
[rima92->device[node001]]% roles
[rima92->device[node001]->roles]% assign provisioning
[rima92->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, the cluster manager 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 RIMA View

The provisioning configuration outlined in `cmsh` mode in section 5.2.2 can be done via RIMA View too, as follows:

A `misc` category can be added via the clickpath
`Grouping`→`Categories`→`Add`→`Settings`→*<name>*
Within the `Settings` tab, the node category should be given a name `misc` (figure 5.3), and saved:

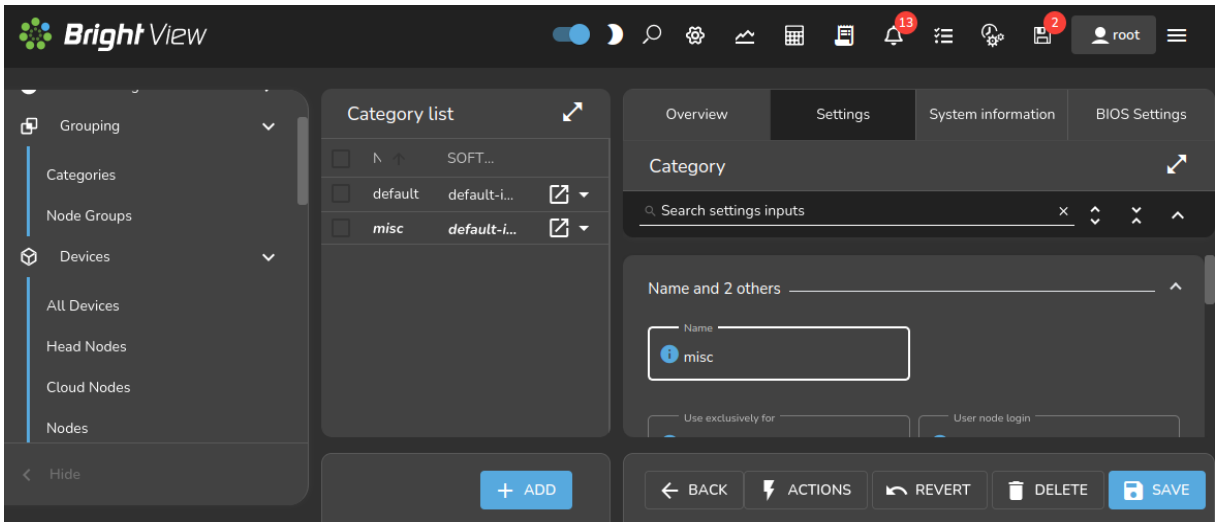


Figure 5.3: RIMA View: Adding A `misc` Category

The `Roles` window can then be opened from within the `JUMP TO` section of the settings pane. To add a role, the `Add` button in the `Roles` window is clicked. A scrollable list of available roles is then displayed, (figure 5.4): The `Roles` window can then be opened from within the `misc` category. To add a role, the `Add` button in the `Roles` window is clicked. A scrollable list of available roles is then displayed, (figure 5.4):

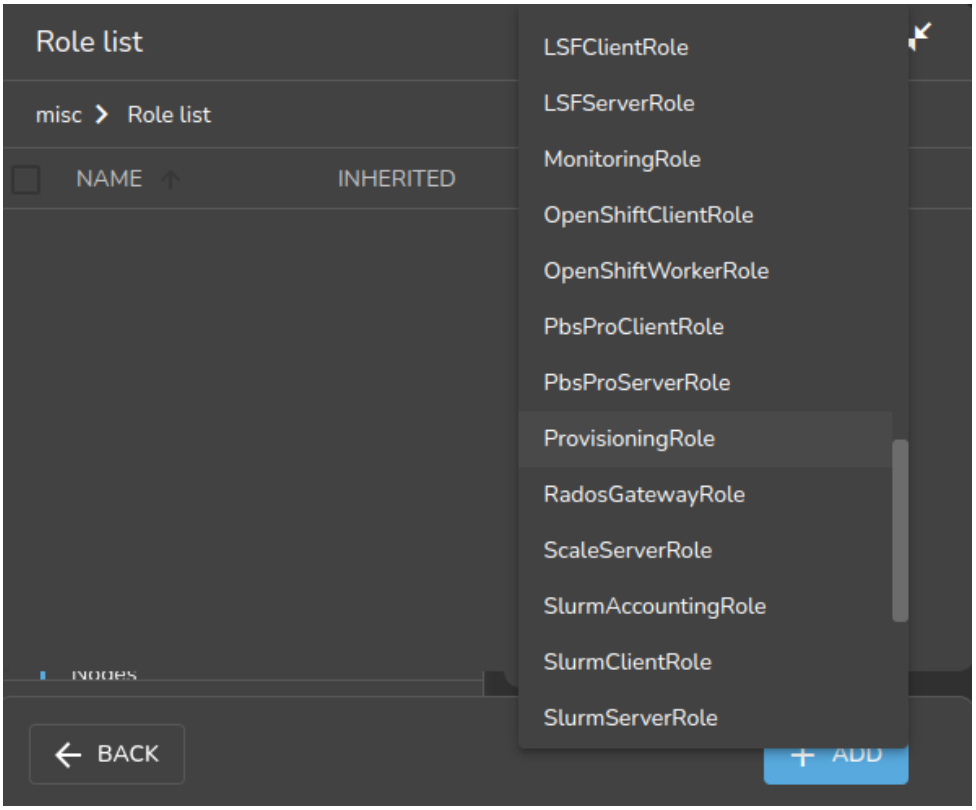


Figure 5.4: RIMA View: Setting A `provisioning` Role

After selecting a role, then navigating via the `Back` buttons to the `Settings` menu of figure 5.3, the role can be saved using the `Save` button there.

The role has properties which can be edited (figure 5.5):

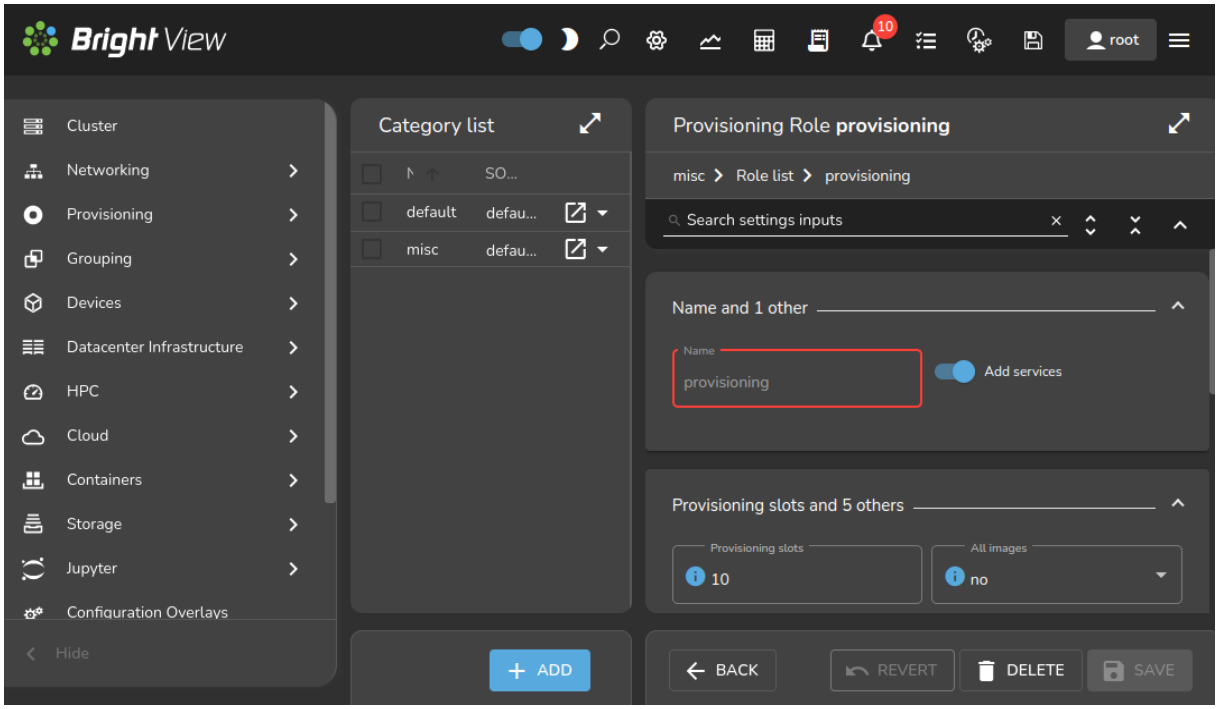


Figure 5.5: RIMA View: Configuring A `provisioning` Role

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 `Save` button of figure 5.5.

The images offered by the provisioning role 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.

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 can be accessed from the softwareimage mode in cmsh. It can also be accessed from RIMA View, via the clickpath Provisioning→Provisioning requests→Update provisioning nodes.

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 (RIMA 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@rima92 ]# cmsh
[rima92]% partition use base
[rima92->partition[base]]% provisioningsettings
[rima92->partition[base]->provisioningsettings]% get autoupdateperiod
[rima92->partition[base]->provisioningsettings]% 86400
[rima92->partition[base]->provisioningsettings]% # is UTC epoch start time modulo 24 hours
[rima92->partition[base]->provisioningsettings]% # set UTC epoch start time modulo 18 hours:
[rima92->partition[base]->provisioningsettings]% set autoupdateperiod 64800
[rima92->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

```
[rima92]% softwareimage updateprovisioners
Provisioning nodes will be updated in the background.

Sun Dec 12 13:45:09 2010 rima92: Starting update of software image(s)\
```

```
 provisioning node(s). (user initiated).
[rima92]% softwareimage updateprovisioners [rima92]%
Sun Dec 12 13:45:41 2010 rima92: Updating image default-image on prov\
isioning node node001.
[rima92]%
Sun Dec 12 13:46:00 2010 rima92: Updating image default-image on prov\
isioning node node001 completed.
Sun Dec 12 13:46:00 2010 rima92: Provisioning node node001 was updated
Sun Dec 12 13:46:00 2010 rima92: 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

```
[rima92->softwareimage]% drain -n master
Nodes drained
[rima92->softwareimage]% provisioningstatus
Provisioning subsystem status
Pending request:            node001, node002
Provisioning node status:
+ rima92
  Slots:                    1 / 10
  State:                    draining
  Active nodes:             node003
  Up to date images:        default-image
[rima92->softwareimage]% provisioningstatus
Provisioning subsystem status
Pending request:            node001, node002
Provisioning node status:
+ rima92
  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

```
[rima92->softwareimage]% drain --role provisioning
...Time passes. Pending
   requests stay in the queue. Then
   admin undrains it...
[rima92->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@rima92 ]# cmsh
[rima92]% partition use base
[rima92->partition[base]]% provisioningsettings
[rima92->partition[base]->provisioningsettings]% get dirtyautoupdatetimeout
[rima92->partition[base]->provisioningsettings]% 300
[rima92->partition[base]->provisioningsettings]% set dirtyautoupdatetimeout 0
[rima92->partition*[base*]->provisioningsettings*]% commit
```

Within RIMA View the parameter is accessible via the clickpath:
`Cluster`→`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 the cluster manager, 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@rima92 ]# cmsh
[rima92]% fspart
[rima92->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 188) 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

```
[rima92->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.

```
[rima92->fspart]% info
Path                          Architecture     OS              Inotify watchers
----------------------------- ---------------- --------------- ----------------
/cm/images/default-image      x86_64           rhel7           0
/cm/images/default-image/boot -                -               0
/cm/node-installer            x86_64           rhel7           0
/cm/shared                    x86_64           rhel7           0
/tftpboot                     -                -               0
/var/spool/cmd/monitoring     -                -               0

[rima92->fspart]% info -s                      (!#with size, takes longer)
Path                          Architecture     OS              Inotify watchers Size
----------------------------- ---------------- --------------- ---------------- ----------------
/cm/images/default-image      x86_64           rhel7           0                4.2 GiB
/cm/images/default-image/boot -                -               0                179 MiB
/cm/node-installer            x86_64           rhel7           0                2.45 GiB
/cm/shared                    x86_64           rhel7           0                2.49 GiB
/tftpboot                     -                -               0                3.3 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

  ```
  [rima92->fspart]% locked
  No locked fsparts
  ```

- The `lock` command prevents a specific fspart from syncing.

  Example

  ```
  [rima92->fspart]% lock /var/spool/cmd/monitoring
  [rima92->fspart]% locked
  /var/spool/cmd/monitoring
  ```

- The `unlock` command unlocks a specific locked fspart again.

  Example

  ```
  [rima92->fspart]% unlock /var/spool/cmd/monitoring
  [rima92->fspart]% locked
  No locked fsparts
  ```

Access to `excludelistsnippets`: The properties of `excludelistsnippets` for a specific fspart can be accessed from the `excludelistsnippets` submode:

Example

```
[rima92->fspart]% excludelistsnippets /tftpboot

[rima92->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

[rima92->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

[rima92->fspart[/tftpboot]->excludelistsnippets]% get default excludelist
# no need for rescue on nodes with a boot role
/rescue
/rescue/*
```

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 7 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 via 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 via 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 the cluster manager, 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 RIMA 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, from a node, or from a category, replacing the original kernel modules list.

Whenever a change is made via 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 RIMA View

In RIMA 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 clickpath that can be followed is:

`Provisioning`→`Software images[default-image]`→`Edit`→`Settings`→`Kernel modules`

which opens up the `Kernel Module list` screen, which allows kernel modules to be removed or added (figure 5.6):

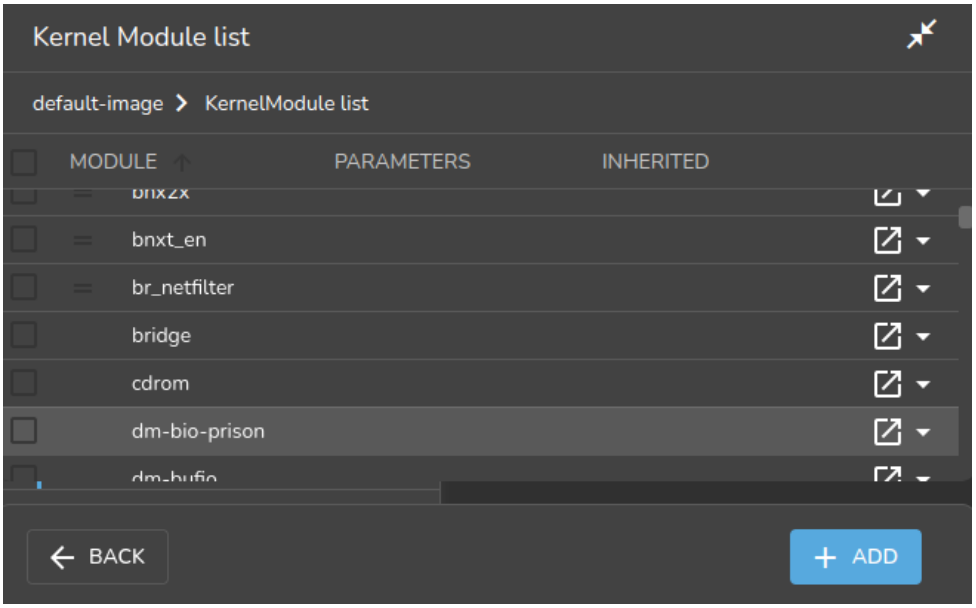


Figure 5.6: RIMA View: Selecting Kernel Modules For Software Images

New kernel modules can be added using the `Add` button, existing kernel modules can be removed using the `Delete` button, and kernel module parameters can be edited using the `Edit` button.

Manually Regenerating A Ramdisk

Regenerating a ramdisk manually via `cmsh` or RIMA 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 7.3). 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 the cluster manager. Not regenerating the cluster manager 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 5.8.5.

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 7.3.4.

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 the cluster manager is done via 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 via 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@rima92 ~]# { service rdma stop; lsmod | cut -f1 -d" "; }>/tmp/a
[root@rima92 ~]# { 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@rima92 ~]# 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 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@rima92 ~]# cmsh
[rima92]% softwareimage use default-image
[rima92->softwareimage[default-image]]% kernelmodules
[rima92...age[default-image]->kernelmodules]% add mlx4_ib
[rima92...age*[default-image*]->kernelmodules*[mlx4_ib*]]% add ib_ipoib
[rima92...age*[default-image*]->kernelmodules*[ib_ipoib*]]% add ib_umad
[rima92...age*[default-image*]->kernelmodules*[ib_umad*]]% commit
[rima92->softwareimage[default-image]->kernelmodules[ib_umad]]%
Tue May 24 03:45:35 2011 rima92: Initial ramdisk for image default-im\
age 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 3.15.2).

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@rima92~]# cmsh
[rima92]% device
[rima92->device]% foreach -n node001..node150 (set provisioninginterface ib0)
[rima92->device*]% commit
```

5.3.4 Omni-Path Provisioning

The Intel Omni-Path Architecture (OPA) can be set up for provisioning in a similar way to the InfiniBand provisioning described in section 5.3.3.

Omni-Path Provisioning: Ramdisk Image Configuration

The hfi1 and ib_ipoib kernel modules must be added to the software image and the initrd image regenerated:

Example

```
[root@rima92 ~]# cmsh
[rima92]% softwareimage use default-image
[rima92->softwareimage[default-image]]% kernelmodules
[rima92...age[default-image]->kernelmodules]% add hfi1
[rima92...age*[default-image*]->kernelmodules*[hfi1*]]% add ib_ipoib
[rima92...age*[default-image*]->kernelmodules*[ib_ipoib*]]% commit
[rima92->softwareimage[default-image]->kernelmodules[ib_ipoib]]%
Thu Jul  4 15:03:35 2019 rima92: Initial ramdisk for image default-im\
age was regenerated successfully.
```

Omni-Path Provisioning: Network Configuration

For the Omni-Path interface on the node, the value of `provisioninginterface` is set to `ib0`, since it still diverges very little from InfiniBand. So, in same way as for InfiniBand before:

Example

```
[root@rima92~]# cmsh
%[rima92]% device
[rima92->device]% foreach -n node001..node150 (set provisioninginterface ib0)
[rima92->device*]% commit
```

5.3.5 VLAN Provisioning

Nodes can be configured for provisioning over a VLAN interface, starting in RIMA Cluster 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@rima92 ~]# ssh node001 "lsmod | grep 8021q"
  [root@rima92 ~]# cmsh
  [rima92]% softwareimage
  [rima92]->softwareimage% use default-image
  [rima92->softwareimage[default-image]]% kernelmodules
  [rima92->softwareimage[default-image]->kernelmodules]% list | grep 8021q
  [rima92->softwareimage[default-image]->kernelmodules]% add 8021q
  [rima92->softwareimage*[default-image*]->kernelmodules*[8021q*]]% commit

  [rima92->softwareimage[default-image]->kernelmodules[8021q]]% device use node001
  [rima92->device[node001]]% reboot
  ...
      some time after boot
  [root@rima92 ~]# 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 7.3.4). An example where the VLANID is appended to some existing parameters could be:

Example

```
[root@rima92 ~]# cmsh
[rima92]% softwareimage use default-image
[rima92->softwareimage[default-image]]% append kernelparameters " VLANID=89"
[rima92->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 7.4.4) 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.4) 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 524.

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 the cluster manager (section 5.4.10), and initializing SELinux if it has been installed and configured (Chapter 7 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 5.7).

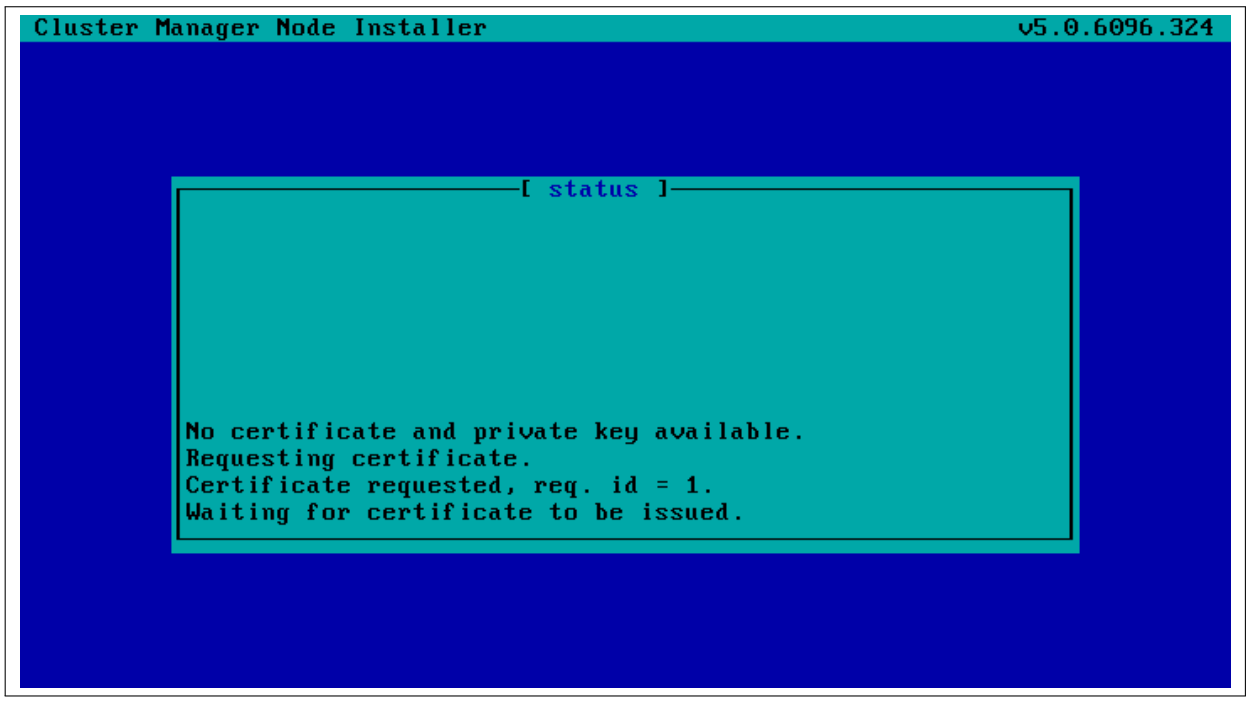


Figure 5.7: Certificate Request

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@rima92 ~]# cmsh
[rima92]% partition
[rima92->partition[base]]% set signinstallercertificates auto
```

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@rima92 ~]# cmsh
[rima92]% network use internalnet
[rima92->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 the cluster manager (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:

```
[rima92->cert]% listrequests
Request ID   Client type  Session ID   Autosign Name
------------ ------------ ------------ -------- ------------------
6            installer    42949672986  No       fa-16-3e-22-cd-13
[rima92->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 via 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 (via 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 5.8). This suggestion can be interrupted, and other node configurations can be selected manually instead through a sub-dialog (figure 5.9). By default (in the main dialog), the original suggestion is accepted after a timeout.

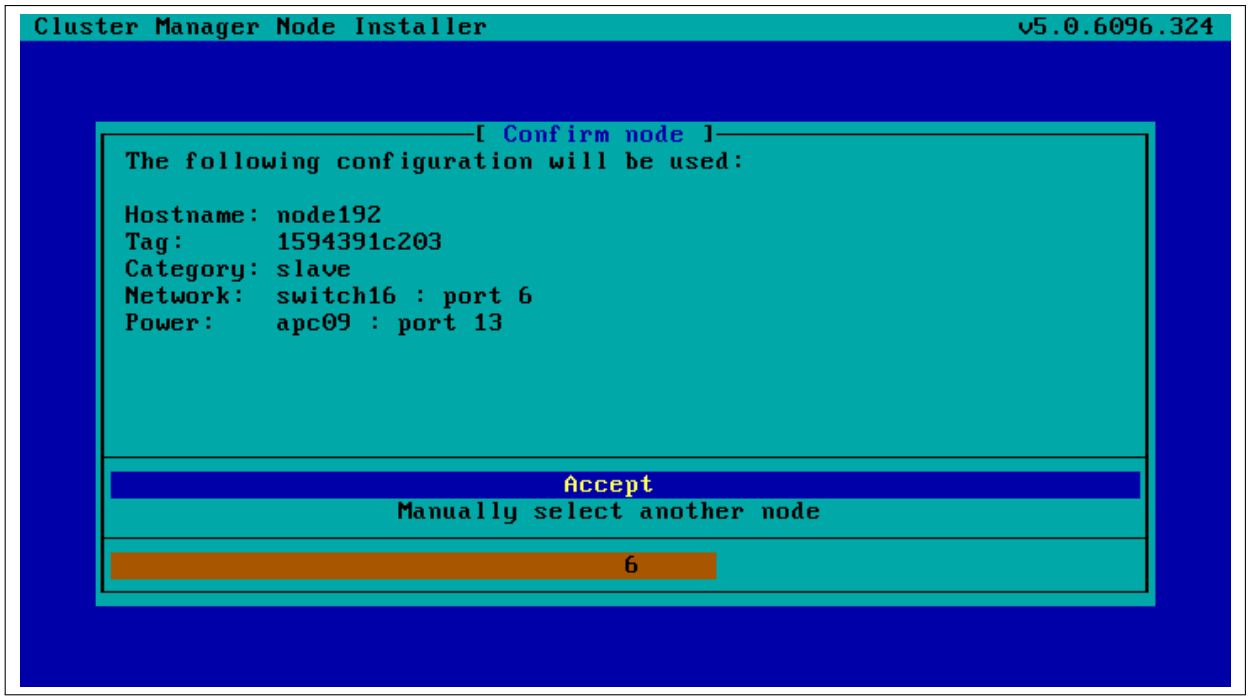


Figure 5.8: Scenarios: Configuration Found, Confirm Node Configuration

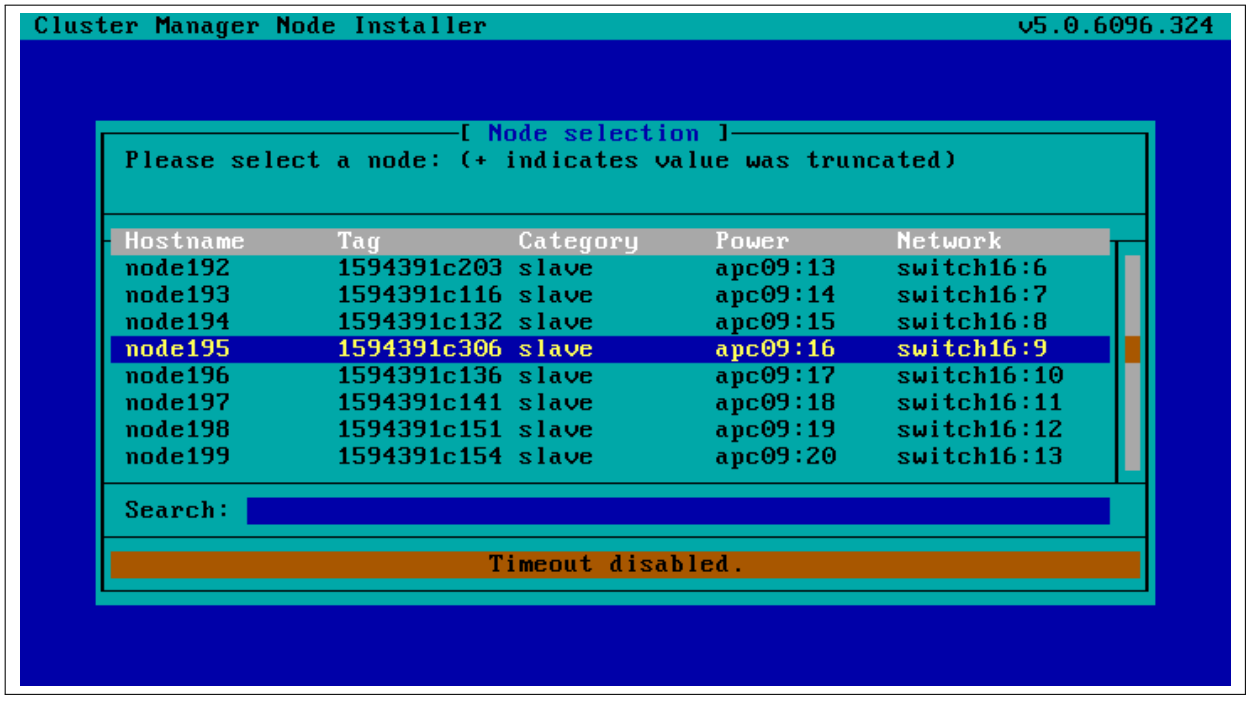


Figure 5.9: Scenarios: Node Selection Sub-Dialog

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 5.10) or to drop into a sub-dialog to manually select a node configuration (figure 5.9). By default, port detection is retried after a timeout.

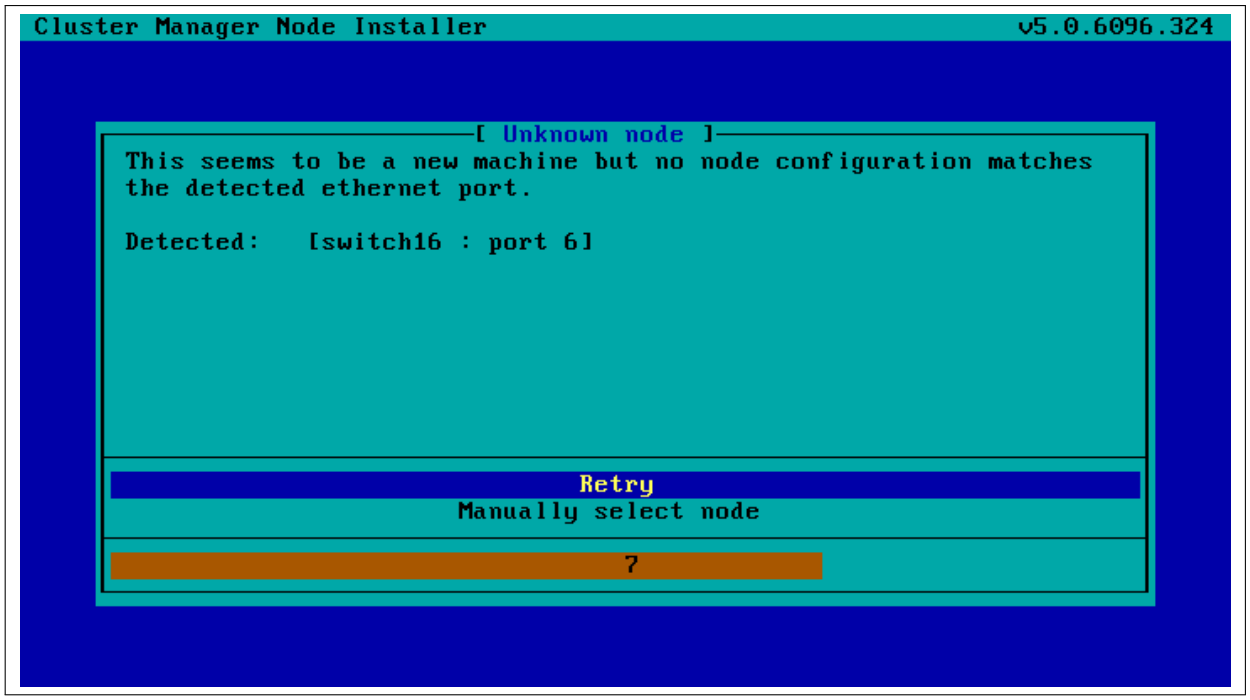


Figure 5.10: Scenarios: Unknown Node, Ethernet Port Detected

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 5.11) or to drop into a sub-dialog to manually select a node configuration (figure 5.9). By default, port detection is retried after a timeout.

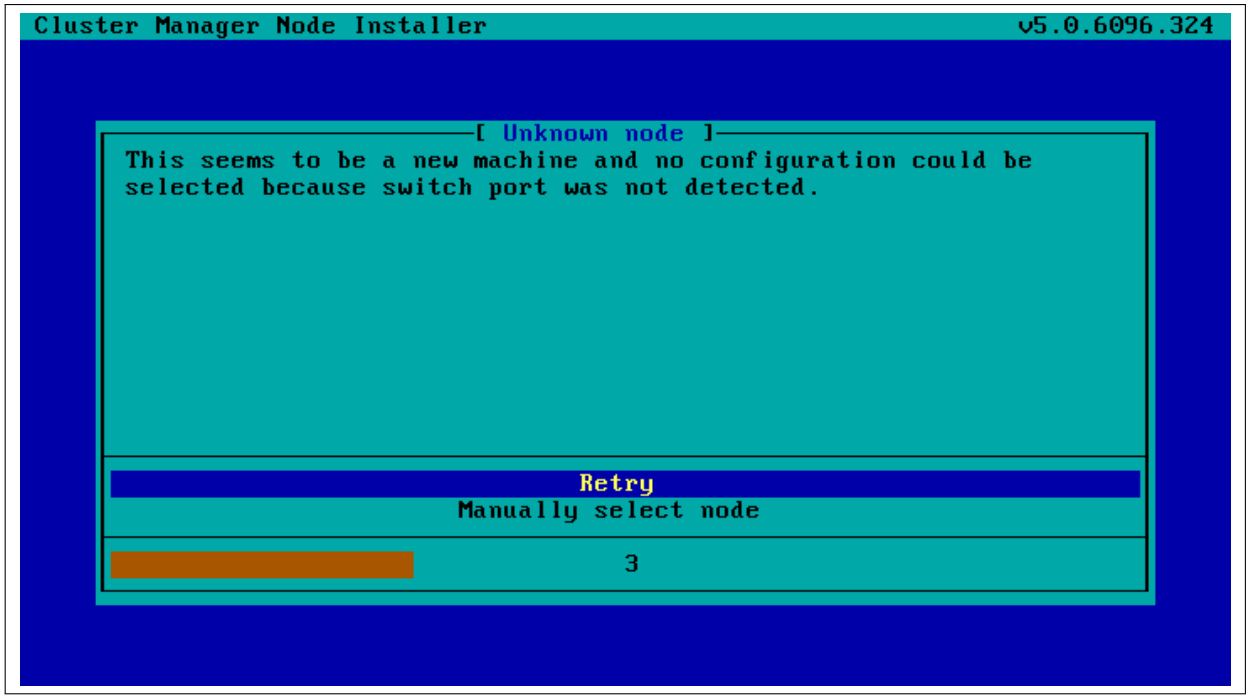


Figure 5.11: Scenarios: Unknown Node, No Ethernet Port Detected

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 5.8). The suggestion can be interrupted, and other node configurations can be selected manually instead through a sub-dialog (figure 5.9). 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 5.12) allowing the user to retry, accept a node configuration that is associated with the detected Ethernet port, or to manually select another node configuration via a sub-dialog (figure 5.9). By default (in the main port mismatch dialog), port detection is retried after a timeout.

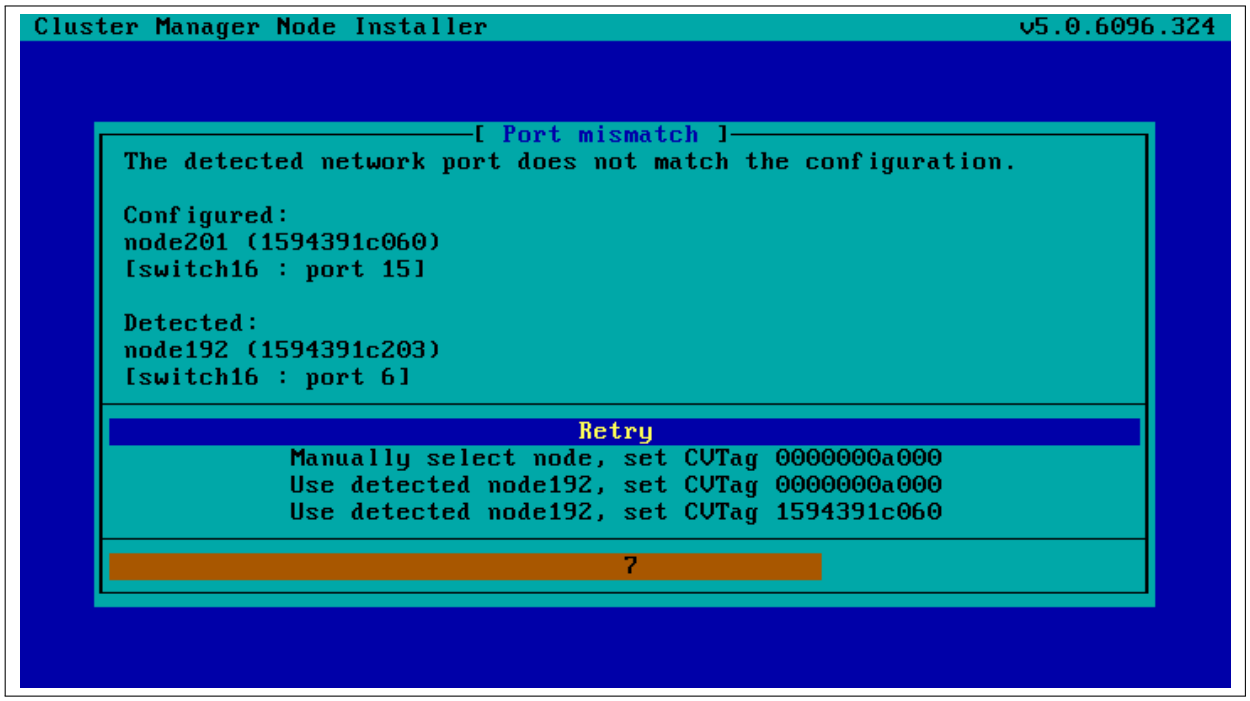


Figure 5.12: Scenarios: Port Mismatch Dialog

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 5.12, 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 5.13). 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.

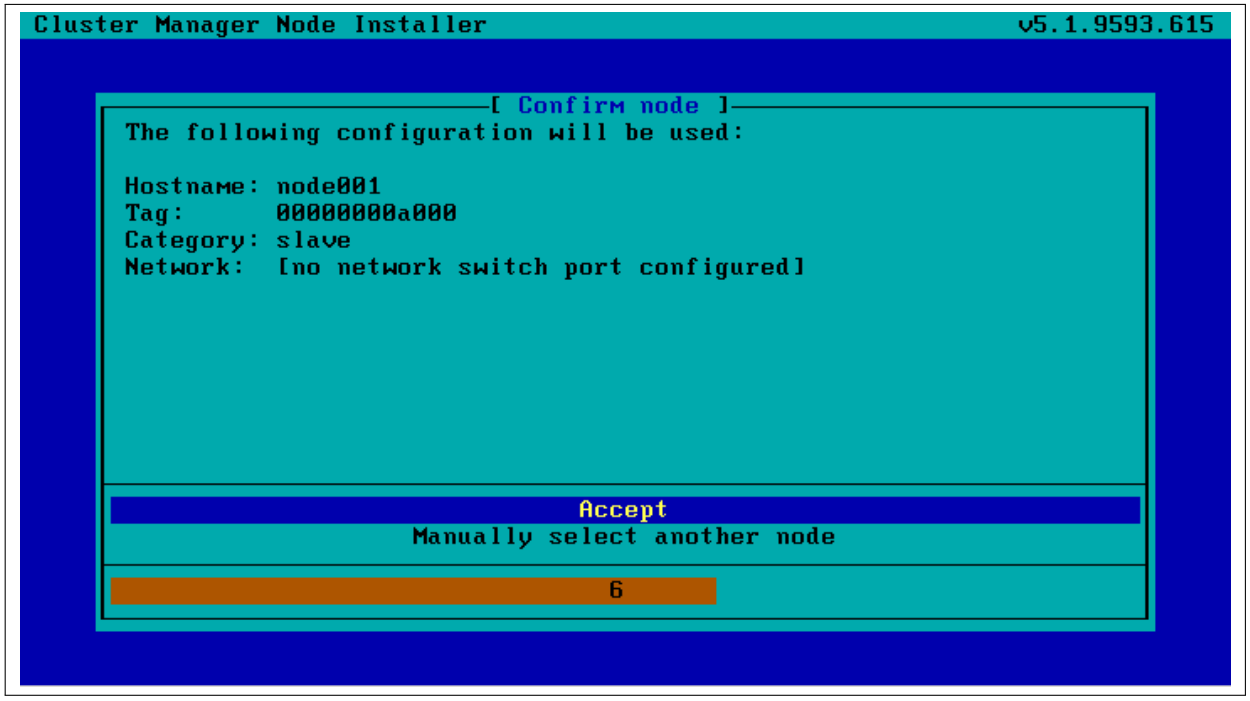


Figure 5.13: Scenarios: Port Unset Dialog

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

```
[rima92->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 3.8.4). After confirming that they are appropriate, the `switchports` property for the specified device can be `set` to the port and switch values.

Example

```
[rima92->device]% showport 00:0C:29:01:0F:F8
switch01:8
[rima92->device]% set node003 switchports switch01:8
[rima92->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 44).

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

  ```
  [rima92->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

```
[rima92->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

```
[rima92->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

```
[rima92->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

```
[rima92->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 RIMA 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 clickpath would be:
`Devices`→`Settings[node001]`→`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

```
[rima92->device]% foreach -n node020..node029 (set rack rack02)
[rima92->device*]% commit
[rima92->device]% foreach -n node030..node039 (set rack rack03)
[rima92->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

```
[rima92->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

```
[rima92->device]% foreach -r rack02 ( clear mac ) ; commit
  [...removes MAC association with nodes from CMDaemon...]

  [...now reboot nodes in rack02 in sequence (not with the cluster manager)...]

[rima92->device]% newnodes

  [...shows sequence as the nodes come up..]

[rima92->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.[1]

This situation can be dealt with from the command line by:

- accepting the node configuration at the regular node console, via a sub-dialog
- accepting the node configuration via cmsh, without needing to be at the regular node console:

  ```
  [rima92->device]% newnodes -s -f -n node022
  ```

Node Identification

The *node identification* resource can be accessed via the clickpath:

Devices→Nodes Identification.

The node identification resource is roughly the RIMA View equivalent to the newnodes command of cmsh, and it opens up the New Node list window (figure 5.14).

As is the case for newnodes in cmsh, the New Node list window of RIMA View 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, New Node list can help assign a node name to the node, assuming the node object exists. After assignment is done, the new status should be saved.

[1] 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

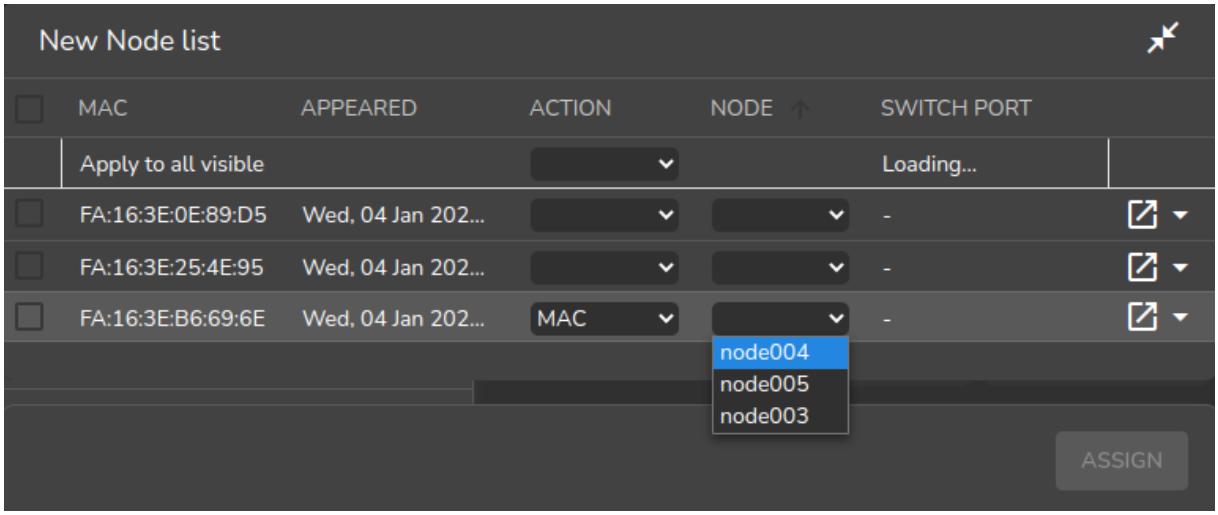


Figure 5.14: Node Identification Resource

The most useful way of using the node 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 RIMA 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 44).

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 clicking on assign automatically assigns the remaining nodes to the next 9 available numbers, say `node328-node336`.

After the assignment, the node-installer looping process on the new nodes notices that the nodes are now known. The node-installer then 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 5.8.5.

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. The cluster manager 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 214) 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 216) 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.[2]

- 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 8 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 216).

[2] 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.

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

```
[rima92]% device foreach -n node001..node999 (set nextinstallmode nosync)
[rima92]% 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 clickpath such as:
   Grouping→Node categories[default]→Edit→Settings→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 5.15). This only affects the current boot. By default the PXE menu install mode option is set to AUTO.



Figure 5.15: 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 a clickpath such as:
Devices→Nodes[node001]→Edit→Settings→Install mode

   It can also be set using cmsh with a one-liner like:

```
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 RIMA View via Devices→Nodes[node001]→Edit→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 RIMA View with a clickpath such as:
   `Grouping→Node categories[default]→Edit→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 a previous point, the install-mode is set by default to `AUTO`.

6. A dialog on the console of the node (figure 5.16) 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`:

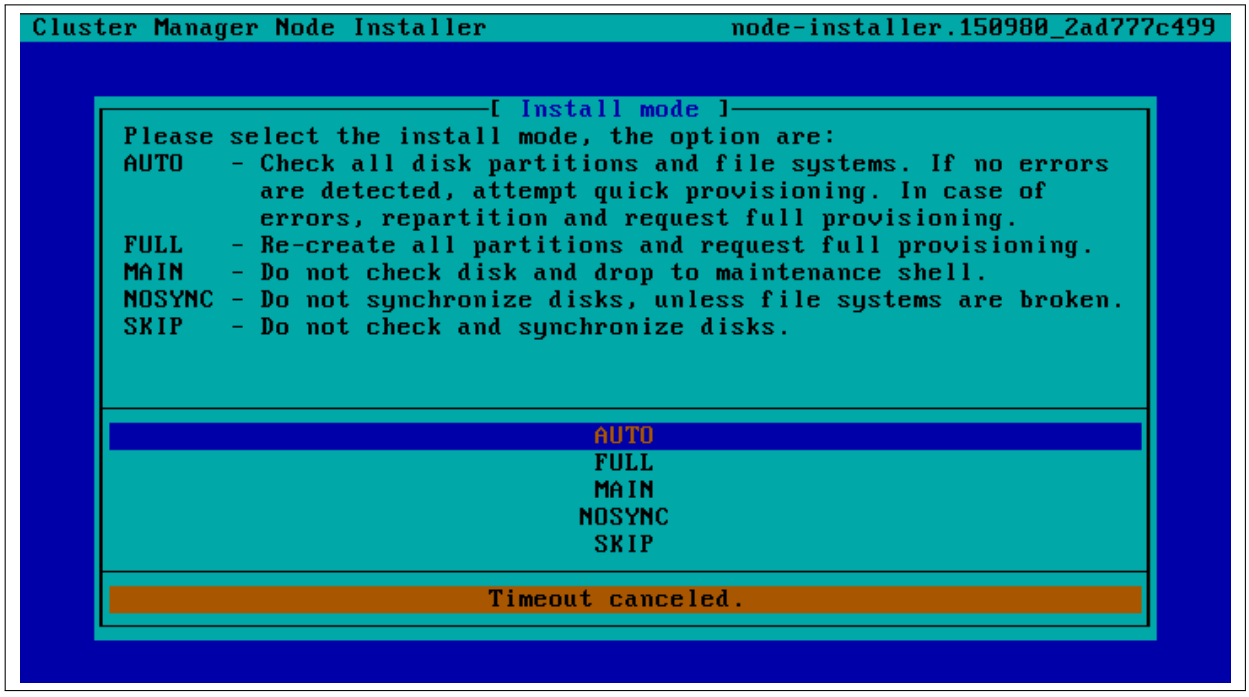


Figure 5.16: Install-mode Setting Option During Node-Installer Run

FULL Install Confirmation via `datanode` Setting

Related to execution mode values is the ability to carry out a FULL install only after explicit confirmation, via 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 set in the node configuration of, for example, `node001` with RIMA View via the clickpath:
`Devices→Nodes[node001]→Edit→Settings→Data node[Yes]`

Alternatively, the parameter `datanode` can be set using a `cmsh` one-liner as follows:

```
[root@rima92 ~]# 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 7.

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 RIMA 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

```
[rima92->device]% installerinteractions -w -n node001 --confirm
Hostname  Action
--------- ------------------------------------------
node001   Requesting FULL Install (partition mismatch)
[rima92->device]%
...07:57:36 [notice] rima92: node001 [ INSTALLER_CALLINGINIT ]...
[rima92->device]%
...07:58:20 [notice] rima92: 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 43).

  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:

```
[rima92->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 3.15.4). 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: The cluster manager 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@rima92 ~]# cmsh
[rima92]% softwareimage list
Name (key)           Path                                    Kernel version               Nodes
-------------------- --------------------------------------- ---------------------------- --------
default-image        /cm/images/default-image                3.10.0-1062.12.1.el7.x86_64  3
[rima92]% softwareimage
[rima92->softwareimage]% lock default-image
[rima92->softwareimage]% islocked
Name           Locked
-------------- --------
default-image  yes
[rima92->softwareimage]% lock default-image
[rima92->softwareimage]% device
[rima92->device]% reboot node001
```

...the cluster administrator makes changes to the node image during a boot, as it waits for the image to unlock...

```
[rima92->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 the cluster manager'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 RIMA View the clickpath is:

Grouping→Node Categories→Edit→Node Category→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 via 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

```
[rima92->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 10), 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 10.1).

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 via 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 10) 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 via ssh from the provisioner to the node being provisioned. This is configured by default in the default cluster manager nodes. However, if a new image is created with the --exclude options for cm-create-image as explained in (section 7.6.2), 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:

```
rima92:~ # cmsh -c "softwareimage provisioningstatus"
Provisioning subsystem status:          idle, accepting requests
Update of provisioning nodes requested: no
Maximum number of nodes provisioning:   10000
Nodes currently provisioning:           0
Nodes waiting to be provisioned:        <none>
Provisioning node rima92:
  Max number of provisioning nodes:     10
  Nodes provisioning:                   0
  Nodes currently being provisioned:    <none>
```

The provisioningstatus command has several options that allow the requests to be tracked. The -r option displays the basic status information on provisioning requests, while the -a option displays all status information on provisioning requests. Both of these options display the request IDs.

The RIMA View equivalent to provisioningstatus is accessed via the clickpath:
Provisioning→Provisioning nodes
By default, it displays basic status information on provisioning requests.

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

```
[rima92->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

```
[rima92->device]% synclog -p node001
/var/spool/cmd/node001-rsync
[rima92->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`).

Aborting Provisioning With `cancelprovisioningrequest`

The `cancelprovisioningrequest` command cancels provisioning.

Its usage is:

```
cancelprovisioningrequest [OPTIONS] [<requestid> ...]
```

To cancel all provisioning requests, it can be run as:

```
rima92:~ # 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:

```
rima92:~ # 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.

The RIMA View equivalents to the `cmsh` versions for managing provisioning requests can be accessed via the clickpath `Provisioning`→`Provisioning Requests`

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, Scientific Linux, CentOS, 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 RIMA View or cmsh (section 3.10.2).

5.4.10 Booting From The Local Hard Drive

By default, a node-installer boots from the software image on the head node via 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 RIMA View, the equivalent is the Install boot record option. This can similarly be enabled and saved in the RIMA 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 the cluster manager 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 7.6.1), 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 7.6.2) 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@rima92 ~]# cmsh
[rima92]% device use node001
[rima92->device[node001]]% get bootloader
syslinux (default)
[rima92->device[node001]]% set bootloader grub
[rima92->device*[node001*]]% commit
```

or, for the `default` category:

```
[root@rima92 ~]# cmsh
[rima92]% category use default
[rima92->category[default]]% get bootloader
syslinux
[rima92->category[default]]% set bootloader grub
[rima92->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 via the boot record on the hard drive.

`BOOTIF` is the default value for the network interface for a node that is configured as a RIMA 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@rima92 ~]# 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 the cluster manager 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

```
[rima92]% device interfaces node001
[rima92->device[node001]->interfaces]% list
Type         Network device name  IP              Network
```

```
------------ ------------------- --------------- ----------------
physical     BOOTIF [prov]       10.141.0.1      internalnet
[rima92->device[node001]->interfaces]% set bootif networkdevicename eth0
[rima92->device*[node001*]->interfaces*]% commit
[rima92->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] rima92: 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 via 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 cluster manager 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 7 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 3.15.4) 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 7.7) 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 309.

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 RIMA View

In the node icons used by RIMA View:

- Nodes in the `UP` state are indicated by an up-arrow.
  - If all health checks (section 8.2.4) 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: ↓

5.2.2 Node States Shown In cmsh

In cmsh, the node state can be found using the status command from device mode for a node:

Example

```
[rima92->device]% status -n node001..node002
node001 ..... [   UP   ] restart-required, health check failed
node002 ..... [  DOWN  ] (hostname changed) restart-required
```

Devices in general can have their states conveniently listed with the `list -f` (page 33) command:

Example

```
[rima92->device]% list -f "hostname:10, status:48"
hostname ( status
---------- ------------------------------------------------
apc01      [   UP   ]
rima92     [   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)`.

5.2.3 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

    ```
    [rima92->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 RIMA 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 RIMA 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 `CLOSED` state can be set from the `device` mode of `cmsh` using the `close` command. 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 RIMA View via the clickpath

  `Devices`→`Nodes`→`Edit`↓`Monitored state`→`Open/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`:

- `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 F.4.1)
- For nodes that have power control[3]:
  - `power -f on`
  - `power -f off`
  - `power -f reset`

In RIMA View, the equivalents for a node `node001` for example, are via the clickpaths:

- `Devices`→`Nodes[node001]`→`Edit`↓`Monitored state`→`Open/Close`
- `Devices`→`Nodes[node001]`→`Edit`↓`Workload`→`Drain/Undrain`
- `Devices`→`Nodes[node001]`→`Edit`↓`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.

[3] power control mechanisms such as PDUs, custom power scripts, and BMCs using IPMI/HP iLO/DRAC/CIMC/Redfish, are described in Chapter 4

- 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 the cluster manager. 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 RIMA 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 RIMA View, it is submitted, for a node node001 for example, using the clickpath:

  Devices→Nodes[node001]→Edit↓Software image→Update node (section 5.6.3).

The imageupdate command and "Update node" menu option 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 (RIMA View), and involves further exclude lists excludelistgrab or excludelistgrabnew. The grabimage command and Grab to image option are covered in detail in section 7.5.2.

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 7.5.2.

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 7.5.2 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` (RIMA 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 5.17.

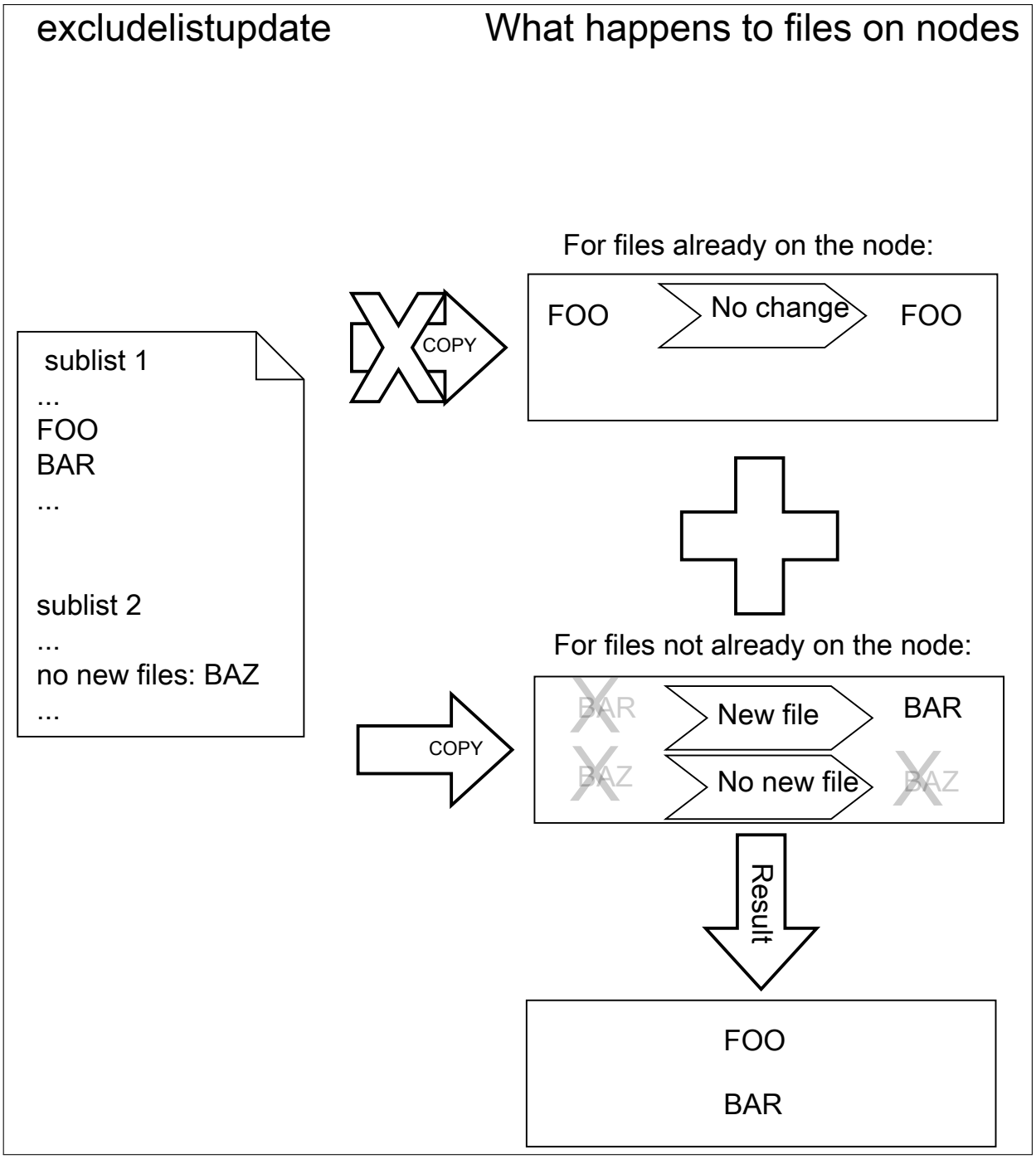


Figure 5.17: Exclude list logic

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 the cluster manager 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 RIMA 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 RIMA View via the clickpath:
`Grouping`→`Node categories`→`Edit`→`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`:

```
[rima92]% device use node001
[rima92->device[node001]]% set excludelistmanipulatescript
(a vi session will start. A script is edited and saved)
[rima92->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:

```
[rima92]% device use node001
[rima92->device[node001]]% get excludelistmanipulatescript
 (the script is put in)
[rima92->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] rima92: Provisioning started: sendi\
ng rima92:/cm/images/default-image to node001:/, mode UPDATE, dry run\
 = yes, no data changes!
[rima92->device[node001]]%
Wed Apr 15 04:55:51 2015 [notice] rima92: Provisioning completed: sen\
t rima92:/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 10.4.8.

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 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 4.4.1 of the *Cloudbursting Manual*. This tool allows additional exclusion to be specified in a simpler way.

The `provisioningassociations` mode: Somewhat related to `excludelistsnippets` is the use of the `provisioningassociations` mode. This is described in section 4.4.2 of the *Cloudbursting Manual*. This mode is used to modify some properties of provisioned file systems.

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

```
[rima92->category[default]]% set excludelistfullinstall
...a text editor such as vi opens up and the list can be edited...
[rima92->category*[default*]]% commit
```

At node level, an exclude list can be set in the same way:

Example

```
[rima92->device[node001]]% set excludelistfullinstall
...a text editor such as vi opens up and the list can be edited...
[rima92->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

```
[rima92->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 232), 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

```
[rima92->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:

```
[rima92->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 232.

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

```
[rima92->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 rima92: Provisioning started on node node001
[rima92->device]% imageupdate -n node001: image update in progress ...
[rima92->device]%
Tue Jan 11 12:13:44 2011 rima92: 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

```
[rima92->device]% imageupdate -n node001 -w
Provisioning started on node node001
node001: image update in progress ...
[rima92->device]% Provisioning completed on node node001
```

5.6.3 Updating Running Nodes: With RIMA View Using the `Update node` Option

In RIMA View, an image update can be carried out by selecting the specific node or category, for example `node001`, and updating it via the clickpath:

`Devices`→`Nodes[node001]`→`Edit`↓`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 RIMA 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, the cluster manager does not allow provisioning if automount (page 535) 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 clickpath of `Devices`→`Nodes[node001]`→`Edit`↓`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 RIMA View Add Functions

Node objects can be added from within the `device` mode of `cmsh` by running the `add` command:

Example

```
[rima92->device]% add physicalnode node002 10.141.0.2
[rima92->device*[node002*]% commit
```

The RIMA View equivalent of this is following the clickpath:
`Devices`→`Nodes`→`ADD`→`PhysicalNode`[Settings]→`Hostname`
then adding the value `node002` to `Hostname`, and saving it.

When adding the node objects in `cmsh` and RIMA 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 via DHCP, while the IP address for the management interface can be set to a static IP address via `cmsh` or RIMA View by the administrator.

Adding new node objects as "placeholders" can also be done from `cmsh` or RIMA 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` button of RIMA View as in the preceding text, there is also a RIMA 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 via the clickpath:

`Devices`→`Nodes`→`CREATE NODES`

This wizard should not be confused with the closely-related node *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 node *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 node identification resource.

In the first screen of the node creation wizard, IP address range suggestions are displayed for the new placeholder nodes. The administrator can override the range. The same screen also allows a category to be selected for the nodes (figure 5.18).

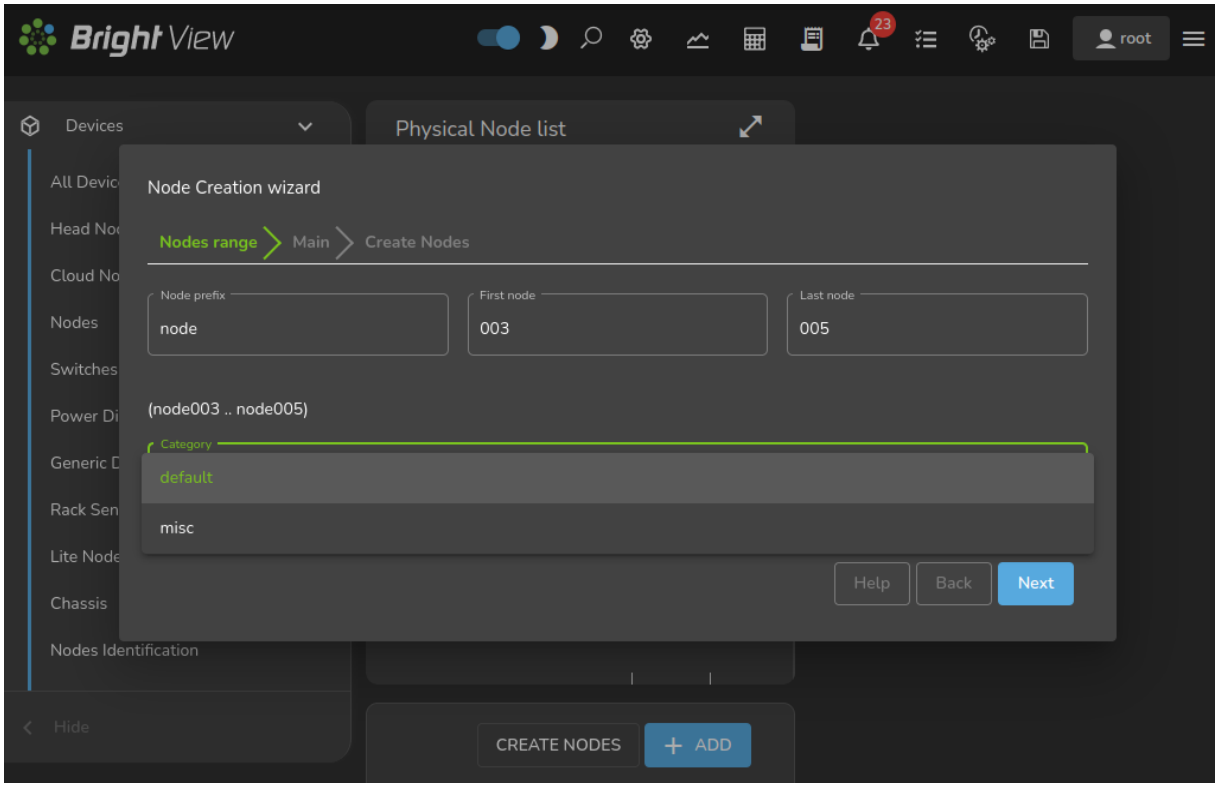


Figure 5.18: Node Creation Wizard: Setting Interfaces

The remaining screens of the wizard configure the interface assignment and excutes the object creation. Once the object has been created, node identification (section 5.4.2) can be carried out.

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 44).

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 7.
  - On the head node, the most relevant software can be updated with `yum`, `zypper`, or `apt`, as explained in section 7.2. 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 7.4. 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@rima92 ~]# 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 RIMA View (figure 3.5, page 69) 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 RIMA 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 RIMA Cluster 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 text:

```
net.ifnames=0 biosdevname=0
```

can be appended to the line starting with `GRUB_CMDLINE_LINUX` in `/etc/default/grub` within the head node. 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.

Example

```
GRUB_CMDLINE_LINUX="rd.lvm.lv=centos/swap vconsole.keymap=us \
crashkernel=auto rd.lvm.lv=centos/root vconsole.font=latarcyr\
heb-sun16 rhgb quiet net.ifnames=0 biosdevname=0"
```

A cautious system administrator may back up the original `grub.cfg` file:

```
[root@rima92 ~]# cp --preserve /boot/grub2/grub.cfg /boot/grub2/grub.cfg.orig
```

The GRUB configuration should be generated with:

```
[root@rima92 ~]# 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

```
[rima92->softwareimage]% list
Name (key)           Path
-------------------- ------------------------------
default-image        /cm/images/default-image
```

```
openstack-image      /cm/images/openstack-image
[rima92->softwareimage]% use default-image
[rima92->softwareimage[default-image]]% append kernelparame\
ters " net.ifnames=0 biosdevname=0"
[rima92->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@rima92 ~]# 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@rima92 ~]# 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 5.3.5) 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 cluster manager 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

```
[rima92->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@rima92 ~]# cmsh -c "softwareimage islocked"
Name           Locked
-------------- --------
default-image  yes
[root@rima92 ~]# cmsh -c "softwareimage unlock default-image"
[root@rima92 ~]# 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 7.7) 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 309.

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 9.7), 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 9.7) 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 9.7.4, 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 222) 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 the rsync transport, logs for node installation are kept under `/var/spool/cmd/`, 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/node002-\.rsync
```

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 7.4), 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 the cluster manager is not aware of this. Running the `createramdisk` command from `cmsh` or the `Recreate Initrd` command via the clickpaths:

- `Devices`→`Nodes`→`Edit`↓`Kernel`→`Recreate Initrd`
- `Grouping`→`Node Categories`→`Edit`↓`Kernel`→`Recreate Initrd`
- `Provisioning`→`Software Images`→`Edit`→`Recreate Initrd`

(section 5.3.2) generates an updated ramdisk for the cluster, and solves the failure for this case.

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 at a wrongly by the administrator in the hierarchy of software image, category, or node (page 192). 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 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 5.19.

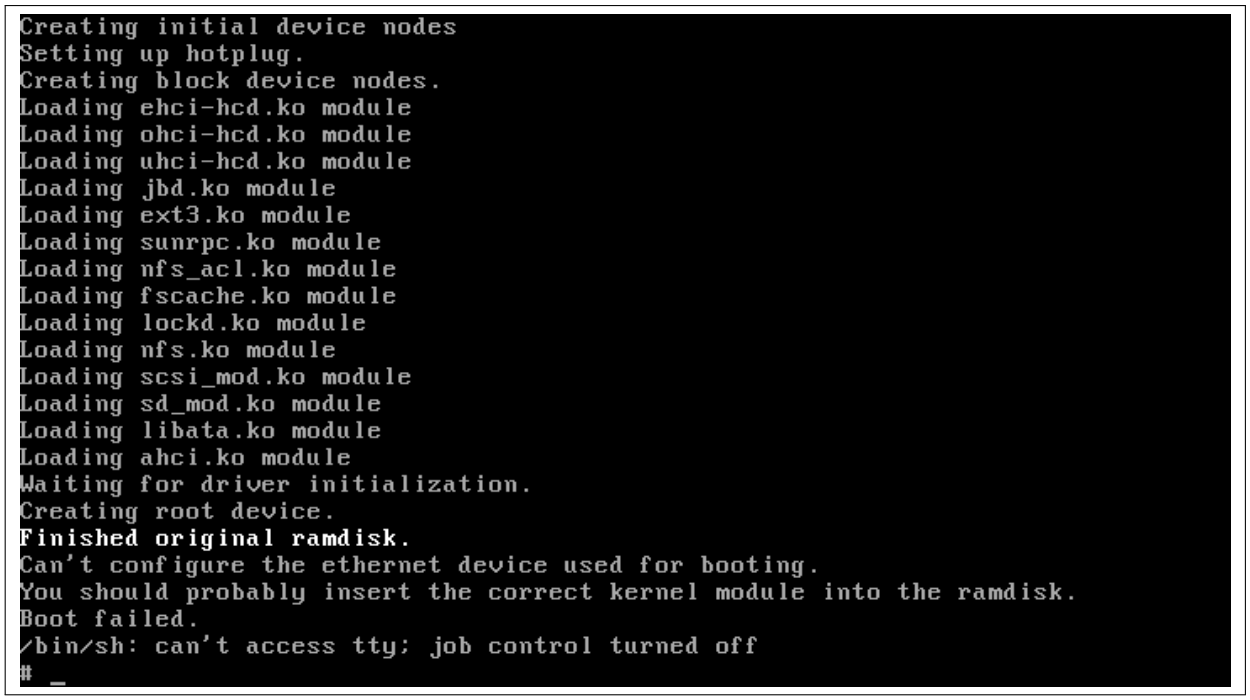


Figure 5.19: No Network Interface

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.

5.8.6 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 5.20. 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.
```

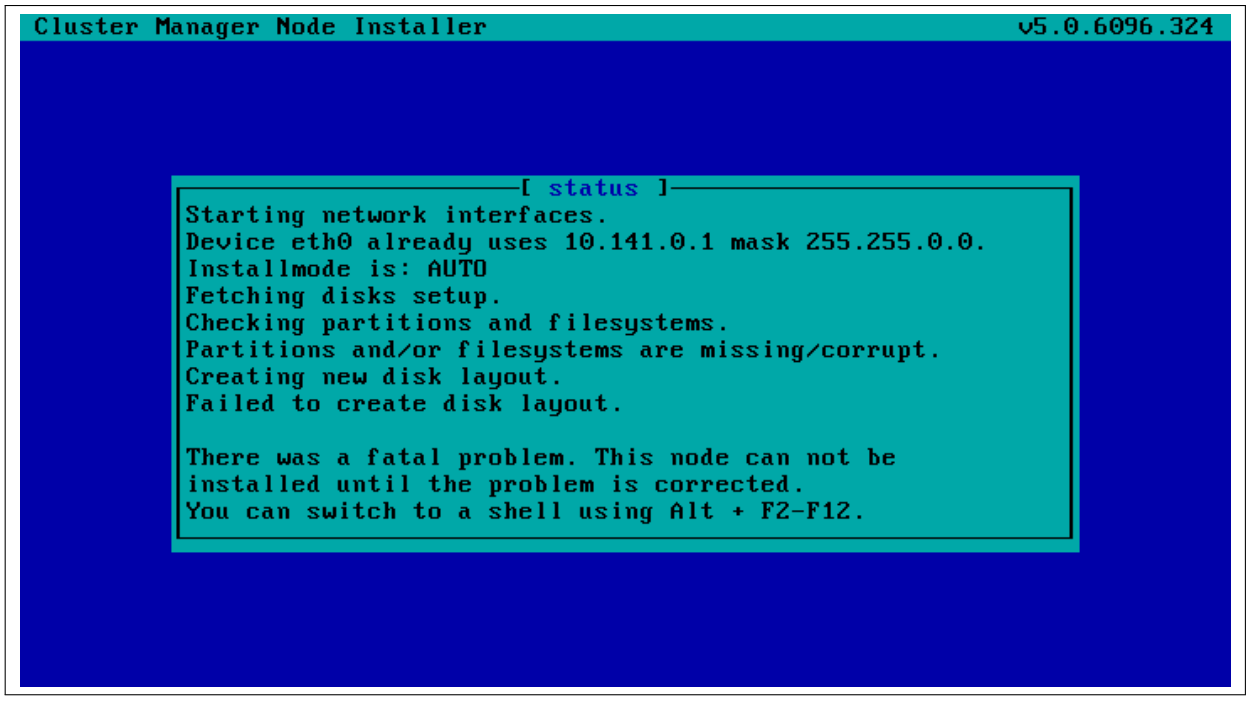


Figure 5.20: No Disk

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 up as a diskless node, to carry out further diagnosis with disk tools such as `smartmontools`.

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

```
[rima92]% softwareimage use default-image
[rima92->softwareimage[default-image]]% kernelmodules
[rima92->softwareimage[default-image]->kernelmodules]% add mpt2sas
[rima92->softwareimage[default-image]->kernelmodules*[mpt2sas*]]% commit
[rima92->softwareimage[default-image]->kernelmodules[mpt2sas]]%
Thu May 19 16:54:52 2011 [notice] rima92: Initial ramdisk for image de\
fault-image is being generated
[rima92->softwareimage[default-image]->kernelmodules[mpt2sas]]%
Thu May 19 16:55:43 2011 [notice] rima92: Initial ramdisk for image de\
fault-image was regenerated successfully.
[rima92->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.

5.8.7 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 5.21.

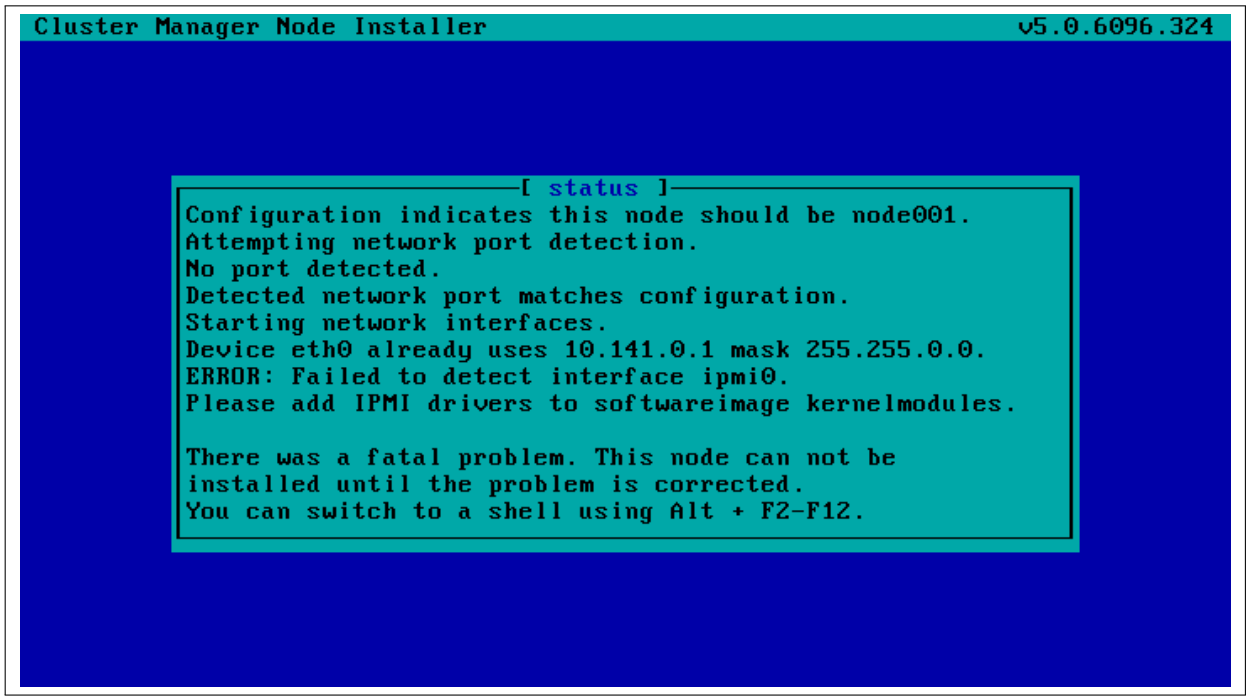


Figure 5.21: No BMC Interface

Usually 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 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` (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 set-

ting `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 the cluster manager 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 multi-arch/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@rima92 ~]# module load ipmitool
[root@rima92 ~]# 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@rima92 ~]# module load ipmitool
[root@rima92 ~]# 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 via the `service ipmi` <*option*> commands, which can show the IPMI service has failed to start up:

Example

```
[root@rima92 ~]# 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@rima92 ~]# 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@rima92 ~]# 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 rima92 systemd[1]: Starting IPMI Driver...
Dec 19 14:34:27 rima92 openipmi-helper[8930]: Startup failed.
Dec 19 14:34:27 rima92 systemd[1]: ipmi.service: main process exited, code=exited, status=1/\
FAILURE
Dec 19 14:34:27 rima92 systemd[1]: Failed to start IPMI Driver.
Dec 19 14:34:27 rima92 systemd[1]: Unit ipmi.service entered failed state.
Dec 19 14:34:27 rima92 systemd[1]: ipmi.service failed.
```

Further details can be found in the journal:

Example

```
[root@rima92 ~]# journalctl -xe | grep -i ipmi
...
-- Unit ipmi.service has begun starting up.
Dec 19 14:34:27 rima92 kernel: ipmi message handler version 39.2
Dec 19 14:34:27 rima92 kernel: IPMI System Interface driver.
Dec 19 14:34:27 rima92 kernel: ipmi_si: Unable to find any System Interface(s)
Dec 19 14:34:27 rima92 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@rima92 ~]# 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 rima92 systemd[1]: Starting IPMI Driver...
Dec 19 14:37:10 rima92 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@rima92 ~]# module load ipmitool
[root@rima92 ~]# 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 F) displayed by the cluster manager 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@rima92 ~]# module load ipmitool
[root@rima92 ~]# 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 the cluster manager, then the changes are automatically shared across the cluster (the single system).

The cluster manager 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 the cluster manager.

Section 6.3 describes how an external LDAP server can be used for authentication services instead of the one provided by the cluster manager.

Section 6.4 discusses how users can be assigned only selected capabilities when using RIMA View or `cmsh`, using profiles with sets of tokens.

6.1 Managing Users And Groups With RIMA View

Within RIMA View:

- users can be managed via the clickpath `Identity Management`→`Users`
- groups can be managed via the clickpath `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 cluster manager: `cmsupport`. This user has no password set by default, which means (section 6.2.2) no logins to this account are allowed by default. The cluster manager 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.

The `+ ADD` button allows users to be added via a `User` parameters window (figure 6.2). The changes in parameter values can be committed via the `SAVE` button in the `User` parameter window.

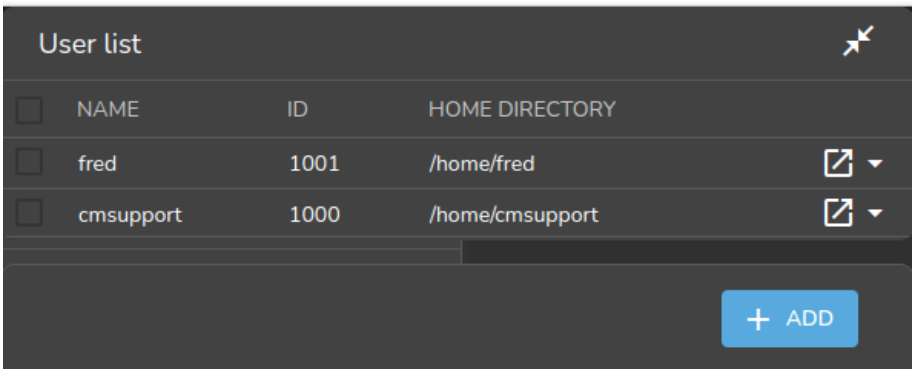


Figure 6.1: RIMA View User Management

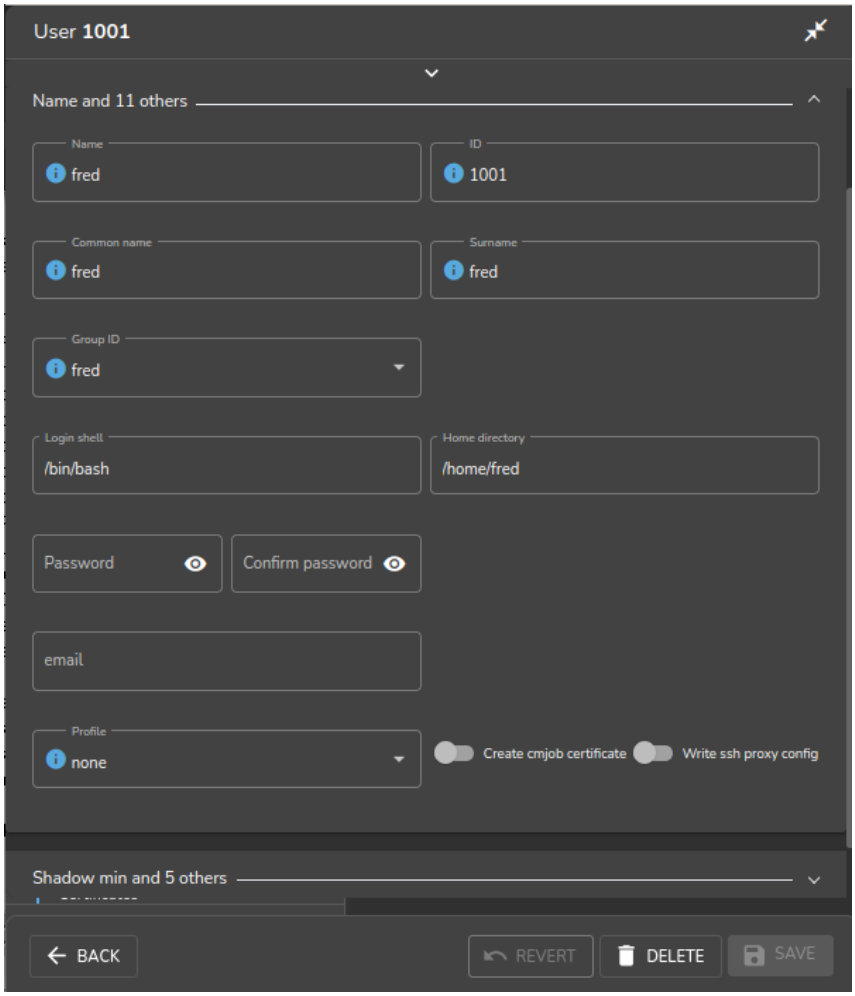


Figure 6.2: RIMA View User Management: Add Dialog

When saving an addition or modification:

- User and group ID numbers are automatically assigned from UID and GID 1000 onward.
- A home directory is created and a login shell is set. Users with unset passwords cannot log in.

Group management in RIMA View is carried out via the clickpath `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 RIMA 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 RIMA View of section 6.1. These functions are run from within the user mode of cmsh:

Example

```
[root@rima92 ~]# cmsh
[rima92]% user
[rima92->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

```
[rima92->user]% add maureen
[rima92->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

```
[rima92->user*[maureen*]]% modified
State  Type                    Name
------ ----------------------- ---------------
+      User                    maureen
```

This corresponds roughly to what is displayed by the Unsaved entities icon in the top right corner of the RIMA View standard display (figure 8.5).

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

```
[rima92->user*[maureen*]]% show
Parameter                      Value
------------------------------ --------------------------
Accounts
Managees
Name                           maureen
Primary group
Revision
Secondary groups
ID
Common name
Surname
Group ID
Login shell
```

```
Home directory
Password                        < not set >
email
Profile
Create cmjob certificate        no
Write ssh proxy config          no
Shadow min                      0
Shadow max                      999999
Shadow warning                  7
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 section 6.2.1 above, 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

```
[rima92->user*[maureen*]]% commit
[rima92->user[maureen]]% show
Parameter                       Value
------------------------------  --------------------------
Accounts
Managees
Name                            maureen
Primary group                   maureen
Revision
Secondary groups
ID                              1001
Common name                     maureen
Surname                         maureen
Group ID                        1001
Login shell                     /bin/bash
Home directory                  /home/maureen
Password                        *********
email
Profile
Create cmjob certificate        no
Write ssh proxy config          no
Shadow min                      0
Shadow max                      999999
Shadow warning                  7
Inactive                        0
Last change                     2021/3/12
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

```
[rima92->user[maureen]]% set

Name:
          set - Set specific user property

Usage:
          set [user] <parameter> <value> [<value> ...]  (type 1)
          set [user] <parameter> [file]                 (type 2)

Arguments:
          user
               name of the user, omit if current is set

Parameters: (type 1)
    commonname ..... Full name (e.g. Donald Duck)
    createcmjobcertificate  Create a certificate with the cloudjob profile for cmjob
    email ............... email
    expirationdate ...... Date on which the user login will be disabled
    groupid ............. Base group of this user
    homedirectory ....... Home directory
    id .................. User ID number
    inactive ............ Number of days of inactivity allowed for the user
    loginshell .......... Login shell
    name ................ User login (e.g. donald)
    password ............ Password
    profile ............. Profile for Authorization
    projectmanager ...... Project manager
    revision ............ Entity revision
    shadowmax ........... Maximum number of days for which the user password remains valid.
    shadowmin ........... Minimum number of days required between password changes
    shadowwarning ....... Number of days of advance warning given to the user before the
```

```
                        user password expires
    surname ............. Surname (e.g. Duck)
    writesshproxyconfig . Write ssh proxy config

Parameters: (type 2)
    notes ............... Administrator notes
[rima92->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

```
[rima92->user[maureen]]% set password seteca5tr0n0my
[rima92->user*[maureen*]]% commit
[rima92->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

```
[rima92->user[maureen]]% clear password; commit
```

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@rima92 ~]#  #first create the LDAP salted SHA-1 hash of the password:
[root@rima92 ~]# /cm/local/apps/openldap/sbin/slappasswd -h {SSHA} -s seteca5tr0n0my
[root@rima92 ~]# {SSHA}sViD+lfSTtlIyOMuGwPGfGd5XKHgEm5d
[root@rima92 ~]# cmsh
[rima92]% user use maureen
[rima92->user[maureen]]% set password
enter new password:          #here and in the next line {SSHA}sViD+lfSTtlIyOMuGwPGfGd5XKHgEm5d is typed in
retype new password:
[rima92->user[maureen]]% commit
[rima92->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.

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

```
[rima92->user[maureen]]% add tim; add fred
[rima92->user*[fred*]]% exit; group;  add printer
[rima92->group*[printer*]]% commit
[rima92->group[printer]]% exit; exit; user
[rima92->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

```
[rima92->user*]% commit
Successfully committed 2 Users
[rima92->user]% group use printer
[rima92->group[printer]]% append members maureen tim fred; commit
[rima92->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:

```
[rima92->group[printer]]% removefrom members tim fred; commit
[rima92->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

```
[rima92->group[printer]]% clear members
[rima92->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 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

```
[rima92->group*[printer*]]% refresh
[rima92->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

```
[rima92->group[printer]]% user use maureen
[rima92->user[maureen]]% remove -d;  commit
Successfully removed 1 Users
Successfully committed 0 Users
[rima92->user]% !ls -d /home/*| grep maureen   #no maureen left behind
[rima92->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 cluster manager LDAP schema.

For RHEL7 and 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, the cluster manager 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 the cluster manager's `ldap` health check object that is used to monitor the LDAP service. Health checks and health check objects are discussed in Chapter 8.

- 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@rima92 ~]# cmsh
[rima92]% user
[rima92->user]% add ldapcheck; commit
[rima92->user[ldapcheck]]% monitoring setup use ldap
[rima92->monitoring->setup[ldap]]% show
Parameter                      Value
------------------------------ ----------------------------------------
...
Arguments
...
[rima92->monitoring->setup[ldap]]% set arguments "ldapcheck"; commit
[rima92->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 RIMA View or `cmsh`.

- RIMA View: the `ldap` health check is disabled via the clickpath:

```
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@rima92 ~]# 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 7.*x* series: the file `/etc/ldap.conf` does not exist. The files in which the changes then 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 `service 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 RHEL7 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 RIMA View or `cmsh`, then altering `cmd.conf` is not required, and the cluster manager's user management capabilities do nothing in any case.
  - If, however, system users and groups are to be managed via RIMA 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 the cluster manager, and so it is very unlikely to match the cluster manager 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 cluster manager 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 cluster manager 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 `service slapd restart` in RHEL 7.*x*.

External LDAP Server Replication: Configuring The Consumer(s)

The consumer is an LDAP server on a cluster manager 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 the cluster manager nodes including the head node(s). If the `/etc/ldap.conf` file does not exist, then the note on page 266 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 `service ldap restart`. 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 the cluster manager 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 266 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 266 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 the cluster manager's high availability system. Configuration changes made on the passive node after the event are therefore persistent.

6.4 Tokens And Profiles

Access to RIMA 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 RIMA 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, or it may be called `powerhandler` if the user is only allowed to carry out power operations. 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 `Profiles` window, accessible via a clickpath of `Identity Management`→`Profiles`

The following preconfigured profiles are available from `cmsh`:

Profile name	Default Tasks Allowed	`nonuser` ?
`admin`	all tasks	no
`bootstrap`	bootstrap tasks	no
`cloudjob`	cloud job submission	yes
`cmhealth`	health-related prejob tasks	yes
`cmpam`	the cluster manager PAM tasks	yes
`litenode`	CMDaemon Lite (section 2.6.7) tasks	yes
`monitoringpush`	pushing raw monitoring data to CMDaemon via a JSON POST (page 408 of the *Developer Manual*)	yes
`node`	node-related tasks, for example by the node-installer	yes
`portal`	user portal viewing	no
`power`	device power	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@rima92 ~]# 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

```
[rima92->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

```
[rima92->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@rima92 ~]# cmsh
[rima92]% user use conner
[rima92->user[conner]]% get profile

[rima92->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.

6.4.1 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@rima92 ~]# cmsh
[rima92]% profile use readonly
[...[readonly]]% removefrom tokens burn_status_token get_burn_log_token
[rima92]%->profile*[readonly*]]% commit
```

Tab-completion after typing in removefrom tokens helps in filling in the tokens that can be removed.

In RIMA View (figure 6.3), the same removal action can be carried out via the clickpath:

Identity Management→Profiles→readonly→Edit→Tokens

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 device subgroup.

The ticks can be removed from the BURN_STATUS_TOKEN and GET_BURN_LOG_TOKEN checkboxes, and the changed settings can then be saved.

6.4.2 Creation Of Custom Certificates With Profiles, For Users Managed By The Cluster Manager's Internal LDAP

Custom profiles can be created to include a custom collection of capabilities in cmsh and RIMA 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 274) or RIMA View (page 275) 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:

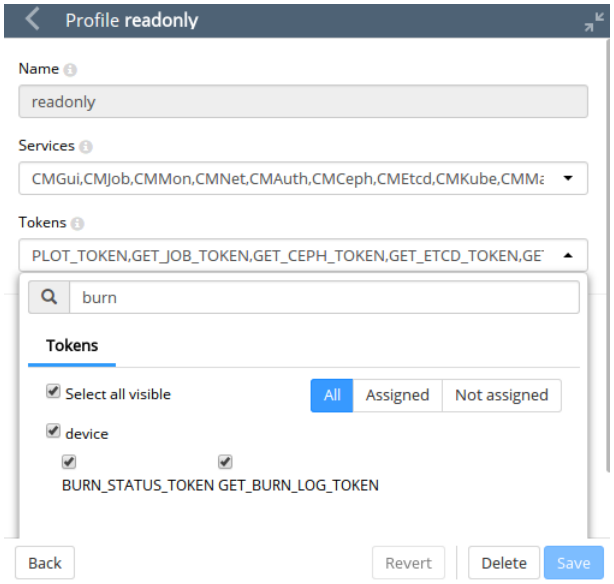


Figure 6.3: RIMA View Profile Token Management

```
[root@rima92]# 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@rima92]#
```

However, using the `openssl` utility for managing certificates is rather inconvenient. The cluster manager 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@rima92 ~]# cmsh
[rima92]% cert
[rima92-> 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 RIMA View: The RIMA View equivalent for listing certificates is via the clickpath `Identity Management`→`Certificates` (figure 6.4):

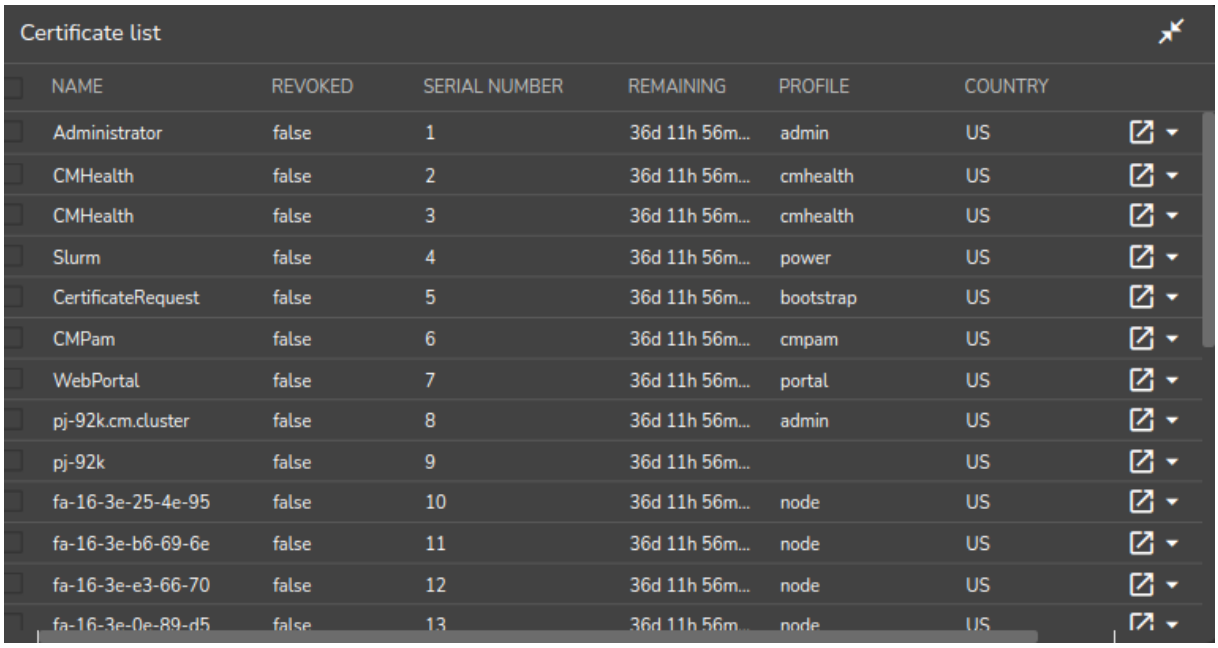


Figure 6.4: RIMA View Certificates List Window

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 that look like:

```
[rima92-> 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
- RIMA View: via the clickpath `Identity Management`→`Users`→`Edit`→`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:

```
[rima92->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

```
[rima92->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] rima92: New certificate request with ID: 16
[rima92->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:

```
[rima92-> 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@rima92 ~]# cd /home/peter
[root@rima92 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@rima92 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@rima92 ~] cmsh -i peterfile.pem -k peterfile.key
[rima92]% 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 RIMA View users: As in the case of `cmsh`, a RIMA 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` from the `Add` or `Edit` dialog 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 RIMA View use this certificate for their RIMA View clients, and are then restricted to the set of tasks allowed by their associated profile.

6.4.3 Creation Of Custom Certificates With Profiles, For Users Managed By An External LDAP

The use of an external LDAP server instead of the cluster manager's for user management is described in section 6.3. Generating a certificate for an external LDAP user must be done explicitly in the cluster manager. 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 the cluster manager.

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 275.

6.4.4 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.

6.4.5 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 572) 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@rima92 ~]# 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 RHEL7 and 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@rima92 ~]# cm-component-certificate -n node001
Sending request to recreate certificates for 1 node to cmd on rima92
[(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 516).

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@rima92 ~]# cm-component-certificate --ca
Sending request to recreate the CA to cmd on rima92
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@rima92 ~]# 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

Post-Installation Software Management

Managing and updating: After RIMA Cluster 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 RIMA Cluster Manager software development team. Managing and updating means carrying out software package management actions such as installation, removal, updating, version checking, and so on.

Package management: Since the cluster manager is built on top of an existing Linux distribution, the administrator should use the distribution-specific package utilities (such as YUM/`rpm` or APT/`dpkg`) 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.

Packages managed by the cluster manager are hosted by the RIMA repository. Access to the RIMA repositories also requires a license (Chapter 4 of the *Installation Manual*). Available packages for a particular cluster manager version and distribution can be viewed via the *package dashboard* at `https://support.brightcomputing.com/packages-dashboard/`.

Software outside of 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.

Upgrading: Upgrading from NVIDIA Bright Cluster Manager version 9.2 to NVIDIA Base Command Manager version 10 is possible, with care, and is covered in the *Upgrade Manual* at `https://support.brightcomputing.com/upgrade-manuals/10.0/upgrade-manual.pdf`.

The software image: 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.

What is covered in this chapter: This chapter covers the techniques of software management for the cluster.

Section 7.1 describes the naming convention for a cluster manager RPM or .deb package.

Section 7.2 describes how an RPM or .deb package is managed for the head node.

Section 7.3 describes how an RPM or .deb kernel package can be managed on a head node or image.

Section 7.4 describes how an RPM or .deb package can be managed on a software image.

Section 7.5 describes how a software other than an RPM or .deb package can be managed on a software image.

Section 7.6 describes how custom software images are created that are completely independent of the existing software image distribution and version.

Section 7.7 describes how multi-architecture and multi-distribution images can be created.

7.1 RIMA Cluster Manager Packages And Their Naming Convention

Like the distributions it runs on top of, the cluster manager uses either .rpm packages, managed by RPM (RPM Package Manager), or .deb (Debian) packages, managed by APT (Advanced Package Tool). For example, the `cmdaemon` package built by the cluster manager has the following .rpm and .deb packages:

```
cmdaemon-9.2-146965_cm9.2_e6f593b676.x86_64.rpm
cmdaemon_9.2-146965-cm9.2-e6f593b676_amd64.deb
```

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* (`9.2`) is the version number of the package
- *revision* (`146965`) is the revision number of the package
- `cm` is used to indicate it is a package built by RIMA for the cluster manager
- *x*.*y* (`9.2`) is the version of RIMA Cluster Manager for which the RPM was built
- *hash* (`e6f593b676`) is a hash, and is only present for the cluster manager packages. It is used for reference by the developers of the cluster manager.
- *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 the cluster manager packages), and the architecture naming convention.

Among the distributions supported by the cluster manager, only Ubuntu uses .deb packages. The rest of the distributions use .rpm packages.

Querying The Packages

To check whether RIMA 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@rima92 ~]# rpm -qf /usr/bin/zless
gzip-1.9-9.el8.x86_64
[root@rima92 ~]# rpm -qf /cm/local/apps/cmd/sbin/cmd
cmdaemon-9.2-146965_cm9.2_e6f593b676.x86_64
```

In the example, `/usr/bin/zless` is supplied by the distribution, while `/cm/local/apps/cmd/sbin/cmd` is supplied by the cluster manager, 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@rima92:~# dpkg -S /cm/local/apps/cmd/etc/cmd.env
cmdaemon: /cm/local/apps/cmd/etc/cmd.env
[root@rima92:~# dpkg -s cmdaemon
Package: cmdaemon
Status: install ok installed
Priority: optional
Section: devel
Installed-Size: 89143
Maintainer: Cluster Manager Development <dev@brightcomputing.com>
Architecture: amd64
Version: 9.2-146965-cm9.2-e6f593b676
Provides: cmdaemon
...
```

As an aside, system administrators should be aware that the RIMA 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 the cluster manager. Replacing the RIMA version with a distribution-provided version can result in subtle and hard-to-trace problems in the cluster, and RIMA cannot offer support 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/`.

7.1.1 The `packages` Command

The cluster manager 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 `cmsh` displays an overview of the installed packages, independent of `rpm` or `deb` package management.

The `-a|--all` option can be used to list all the packages installed on a particular node:

Example

```
[rima92]% device use node001
[rima92->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

```
[rima92]% device
[rima92->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

```
[rima92]% device
[rima92->device]% packages -c default -f cmdaemon
Node     Type     Name      Version                       Arch     Size     Install date
-------- -------- --------- ---------------------------- -------- -------- --------------------
node001  deb      cmdaemon  9.2-151702-cm9.2-3edc085657  amd64    76MiB    2022/10/27 12:19:18
node002  deb      cmdaemon  9.2-151702-cm9.2-3edc085657  amd64    76MiB    2022/10/27 12:19:18
...
```

Further options, and examples, can be listed by running the `help packages` command within the `device` mode of `cmsh`.

7.2 Managing Packages On The Head Node

7.2.1 Managing RPM Or .deb Packages On The Head Node

Once the cluster manager has been installed, distribution packages and the cluster manager 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 the cluster manager supports. YUM is not set up by default in SUSE, and it is better not to install and use it with SUSE 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 cluster manager 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.

RIMA maintains YUM and zypper repositories of its packages at:

```
http://updates.rima.com/yum
```

and updates are fetched by YUM and zypper for the cluster manager packages from there by default, to overwrite older package versions by default.

For Ubuntu, the RIMA .deb package repositories are at:

```
http://updates.rima.com/deb
```

Accessing the repositories manually (i.e. not using `yum`, `zypper`, or `apt`) requires a username and password. Authentication credentials are provided upon request. For more information on this, `supportteam@rima.com` should be contacted.

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 the cluster manager installations from receiving malicious updates, all the cluster manager packages are signed with the RIMA GPG public key (0x5D849C16), installed by default in /etc/pki/rpm-gpg/RPM-GPG-KEY-cm for Red Hat and derivaties. The RIMA 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 RIMA 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 cluster manager keyring is already imported into /etc/apt/trusted.gpg.d/brightcomputing-archive-cm.gpg if the cm-config-apt package, provided by the RIMA repository, has been installed. The cm-config-apt package is installed by default for the Ubuntu edition of the cluster manager.

Third Party Packages

Installation of the following third party packages, most of which are repackaged by RIMA for installation purposes, is described in Chapter 7 of the *Installation Manual*. These are installed mostly on the head node, but some packages work as a server and client process, with the clients installed and running on the regular nodes:

Most of the third party packages in the following list are repackaged by RIMA for installation purposes. These are installed mostly on the head node, but some packages work as a server and client process, with the clients installed and running on the regular nodes. The packages are described in Chapter 7 of the *Installation Manual*:

- Modules (section 7.1)
- Shorewall (section 7.2)
- Compilers (section 7.3):
  - GCC (section 7.3.1)
  - Intel Compiler Suite (section 7.3.2)
- CUDA (section 7.4)

Exclusion of packages on the head node can be carried out as explained in section 7.3.2, where the kernel package is used as an example for exclusion.

7.2.2 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 by RIMA or by 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
```

7.3 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.

7.3.1 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 7.2).
- 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 7.4), but with some special aspects that are discussed in this section.

When a kernel is updated or reinstalled (section 7.3.3), kernel-specific drivers, such as OFED drivers may need to be updated or reinstalled. OFED driver installation details are given in section 7.6 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 7.4, extra considerations for kernel packages are:

- The kernel must also be explicitly set in CMDaemon (section 7.3.3) 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, 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 7.4.3).
- 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.

7.3.2 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 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@rima92 ~]# yum repolist
...
122 packages excluded due to repository priority protections
repo id                    repo name                                          status
base/7/x86_64              CentOS-7 - Base                                    10,067+30
cm-rhel7-9.2/x86_64        CM 9.2 for Red Hat Enterprise Linux 7              10,949+56
epel/x86_64                Extra Packages for Enterprise Linux 7 - x86_64     13,324+92
extras/7/x86_64            CentOS-7 - Extras                                      301+3
updates/7/x86_64           CentOS-7 - Updates                                     332
repolist: 34,973
[root@rima92 ~]# 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 (1) add-apt-repository`) edits the repository sources by line. Running `add-apt-repository -h` shows options and examples.

7.3.3 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 7.4), 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 (section 7.6 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 cluster manager cluster, this default distribution value is overridden by a default cluster manager 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 RIMA View or `cmsh`.

- To enable it using RIMA View, the `Kernel version` entry for the software image should be set. This can be accessed via the clickpath `Provisioning`→`Software images`→`Edit`→`Settings`→`Kernel version` (figure 7.1).

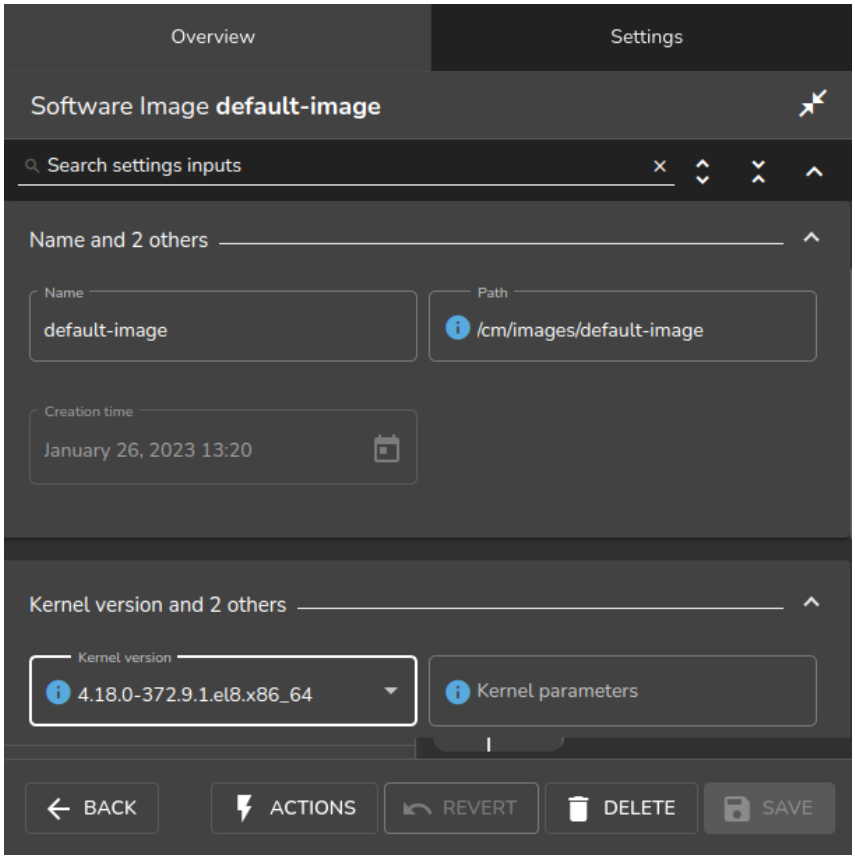


Figure 7.1: Updating A Software Image Kernel With RIMA View

- 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@rima92 ~]# cmsh
[rima92]% softwareimage
[rima92]->softwareimage% use default-image
[rima92->softwareimage[default-image]]% set kernelversion  3.10.0-327.3.1.el7.x86_64
[rima92->softwareimage*[default-image*]]% commit -w
```

Tab-completion suggestions for the `set kernelversion` command will display the available values for the kernel version.

7.3.4 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 RIMA View, to enable booting with this kernel option setting, the clickpath `Provisioning`→`Software images`→`Edit`→`Settings`→`Kernel parameters` (figure 7.1) 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@rima92 ~]# cmsh
[rima92]% softwareimage
[rima92]->softwareimage% use default-image
[rima92->softwareimage[default-image]]% append kernelparameters " apm=off"
[rima92->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.

7.3.5 Kernel Driver Modules

RIMA 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 RIMA 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 cluster manager 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 distributions provide Intel wired Ethernet driver modules as part of the kernel they provide. RIMA 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`" button (section 7.5.2), this will write 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 7.3.5).

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 the cluster manager. The details of how this is done is covered in the CUDA software section (section 7.4 of the *Installation Manual*).

Kernel Driver Modules: OFED And OPA Stack Installation

By default, the distribution provides the OFED stack used by the kernel to manage the InfiniBand or RDMA interconnect. Installing a cluster manager repository OFED stack to replace the distribution version is covered in section 7.6 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.

Installing a cluster manager repository Omni-Path (OPA) stack is covered in section 7.7 of the *Installation Manual*.

7.4 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.

7.4.1 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 the cluster manager is to use a wrapper provided by the cluster manager, `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 RHEL7.4, the command in the preceding example would have `--releasever=7.4` 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@rima92 ~]# chroot /cm/images/default-image
[root@rima92 /]# yum update    #for RHEL variants
...updates happen...
[root@rima92 /]# 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

```
rima92:~# chroot /cm/images/default-image
rima92.cm.cluster:/ # zypper up            #for SLES
...updates happen...
rima92.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 cluster manager 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@rima92:~# 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@rima92:/# apt update; apt upgrade #for Ubuntu
...An upgrade session runs in the image root. Some administrator inputs may be needed...
root@rima92:/# 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 7.3.2.

- 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, and only want the cluster manager 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@rima92 ~]# cd /cm/images/default-image/etc/yum.repos.d/
[root@rima92 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.

7.4.2 Installing From Head Into The Image: Updating The Node

If the images are in place, then the node that use those images do not 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 (section 5.6), if no reboot is required by the underlying Linux distribution.

7.4.3 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@rima92 ~]# cmsh -c "softwareimage lock default-image"
[root@rima92 ~]# yum --installroot /cm/images/default-image update
[root@rima92 ~]# 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 Scientific Linux 7 head node to a RHEL8 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.

  RIMA'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 7.3.3) can be inspected using the cluster manager with:

  Example

  ```
  cmsh -c "softwareimage; use default-image; get kernelversion"
  ```

7.4.4 Managing A Package In The Node-Installer Image

A special software image is the node-installer image. The node-installer image was introduced in RIMA Cluster Manager version 9.0, to make multiarch (section 7.7) 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.

7.5 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 7.5.1)
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 the cluster manager's special utilities (section 7.5.2)

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 7.6.

However, this current section (7.5) 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 7.2.2, appropriate software components go under:

```
/cm/shared/
|-- apps
|-- docs
|-- examples
|-- licenses
`-- modulefiles
```

7.5.1 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 7.4:

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.

7.5.2 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 7.5.1, 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 RIMA 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 7.5.1).
- 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 RIMA View

In RIMA 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 via: `Devices`→`Nodes[node001]`→`Actions`→`Software image`→`Grab to image`→*options* (figures 7.2 and 7.3):

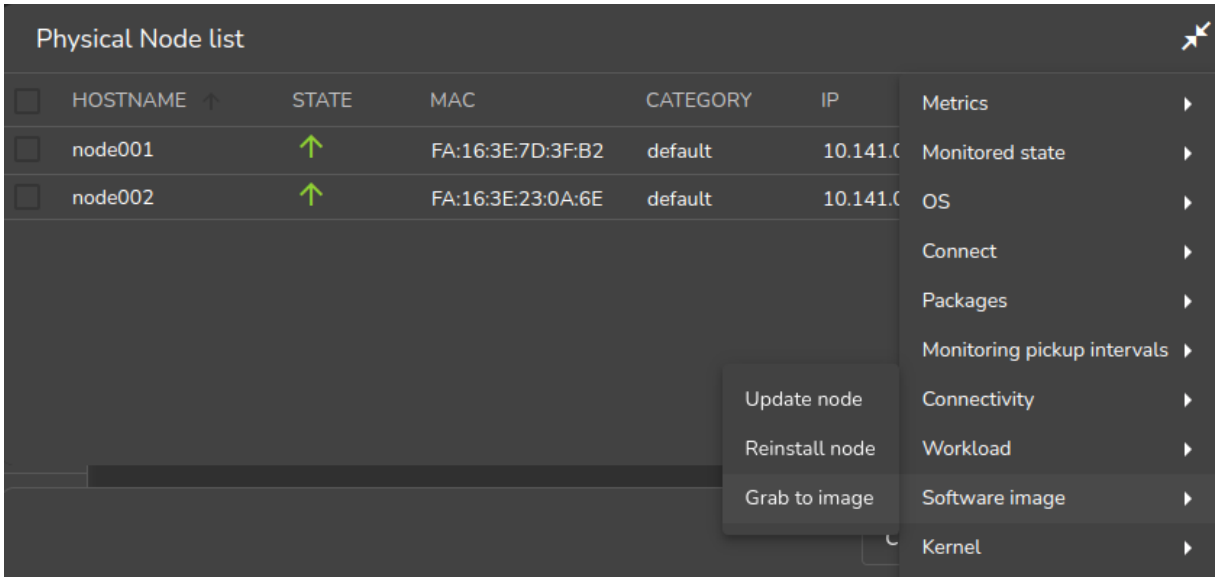


Figure 7.2: Synchronizing node-to-image: accessing `Grab to image`

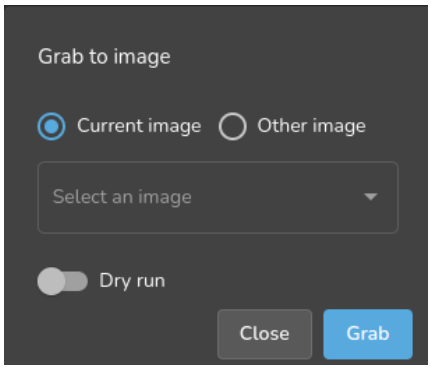


Figure 7.3: Synchronizing node-to-image: screen options for `Grab to image`

In the `Grab to image` screen (figure 7.3):

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` block, navigated to via `Grouping`→`Node categories[default-image]`→`Edit`→`Settings`→`Exclude list grab`. This exclude list is known as `excludelistgrab` (page 299) in `cmsh`.

2. `Other 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 are specified in the `Exclude list grab new` block, navigated to via `Grouping`→`Node categories[default-image]`→`Settings`→`Exclude list grab new`. This exclude list is known as `excludelistgrabnew` (page 299) 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 via the clickpath `Provisioning`→`Software images`↓`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 clickpath `Grouping`→`Categories`↓`Clone`, and setting a name for the new category, for example `newcategory`. The old image in `newcategory` is changed to the new image `newimage` via the clickpath `Grouping`→`Categories`→`Edit`→`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 clickpath for this is `Devices`→`Nodes`→`Edit`→`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 RIMA 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 the `Other image` option 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@rima92 ~]# cmsh
[rima92]% softwareimage
[rima92->softwareimage]% clone default-image default-image1
[rima92->softwareimage*[default-image1]]% commit
[rima92->softwareimage[default-image1]]% category
[rima92->category]% clone default default1
[rima92->category*[default1*]]% commit
[rima92->category[default1]]% set softwareimage default-image1
[rima92->category*[default1*]]% commit
[rima92->category[default1]]% device
```

```
[rima92->device]% grabimage -w -i default-image1 node001
[rima92->device]%
Mon Jul 18 16:13:00 2011 [notice] rima92: Provisioning started on node node001
[rima92->device]%
Mon Jul 18 16:13:04 2011 [notice] rima92: 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 `grabimage` command run with the `-i` option. The list can be accessed and edited via

  `cmsh>category[`*image*`]>set excludelistgrabnew`

  It corresponds to the `Exclude list grab new` exclusion list associated with `Grab to image` when using the `Other image` option (figure 7.3) in RIMA View.

- `excludelistgrab`: This is used with the `grabimage` command, run without the `-i` option. The list can be accessed and edited via

  `cmsh>category[`*image*`]>set excludelistgrab`

  It corresponds to the `Exclude list image grab` exclusion list associated with `Grab to image` with the `Current image` option (figure 7.3) in RIMA View.

7.6 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 the cluster manager.

Creating a custom software image consists of two steps. The first step (section 7.6.1) is to create a *base (distribution) archive* from an installed base host. The second step (section 7.6.2) 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 7.7.1), which is a wrapper to `cm-create-image`.

7.6.1 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.

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 7.6.2.

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 --acls --xattrs \
--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 --acls --xattrs --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/
```

The `--acls` and `--xattrs` options are not explicitly required for `rsync` with SLES12 and beyond. They are also not required for RHEL7 and beyond, and derivatives.

SELinux and file attributes: To use SELinux on compute nodes for RHEL7 and its derivatives, extended attributes must not be used.

The defaults can be modified, if needed, by adjusting attributes for partitions via `cmsh`, in `fspart` mode:

Example

```
[rima92->fspart]% foreach * (set rsyncxattr no)
[rima92->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.

7.6.2 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 303).

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 7.2) 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 Novell (section 5.2 of the *Installation Manual*). An offline repository can be constructed as described in section 7.6.3 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
               (-a FROMARCHIVE | -d FROMDIR | -h FROMHOST | -k | --frombfb FROMBFB)
               [--add-only] [--arch IMAGE_ARCH] [--os IMAGE_OS] [-c CMREPO | --cmdvd CMDVD]
               [-b BASEDISTREPO] [-e] [-f] [-g ENABLEEXTRAREPO] [-i IMAGEDIR] [-j EXCLUDEDIST]
               [--holdpackages HOLDPACKAGES] [-l RESOLVCONF] [-m] [-n IMAGENAME]
               [-o EXCLUDE_FROM] [-q EXCLUDEHWVENDOR] [-r] [-s] [-t {node-installer,cmshared}]
               [-u] [-v] [-w HWVENDOR] [-x EXCLUDECM] [-y] [-z CUSTOM_PRE_INSTALL_SCRIPT]
               [--no-progress] [-L LOGFILE] [--sles-allow-vendor-change]
               [--skip-connectivity-check] [--tar-options ...]
```

Examples In Usage Of `cm-create-image`

Explanations of the usage text follow:

1. In the following, a base distribution archive file, `/tmp/RHEL7.tar.gz`, is written out to a software image named `rhel7-image`:

   ```
   cm-create-image --fromarchive /tmp/RHEL7.tar.gz --imagename rhel7-image
   ```

   The image with the name `rhel7-image` is created in the CMDaemon database, making it available for use by `cmsh` and RIMA View. If an image with the above name already exists, then `/cm/create-image` will exit and advise 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/rhel7-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/RHEL7.tar.gz` is written out to the `/cm/images/test-image` directory. The software image is given the name `rhel7-image`:

   ```
   cm-create-image --fromarchive /tmp/RHEL7.tar.gz --imagename rhel7-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/SLES11-image`, and it is that directory which is then used to place the image under a directory named `/cm/images/sles11-image/`. Also, the software image is given the name `sles11-image`:

   ```
   cm-create-image --fromdir /cm/images/SLES11-image --imagename sles11-image
   ```

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 7.5.2) in `cmsh`. It requires passwordless access to the node in order to work. Generic nodes, that is nodes that are not managed by the cluster manager, 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> ...
   ```

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 the cluster manager, and is the base-distribution set of packages, which may not work with some features of the distribution or the cluster manager. 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 the cluster manager. For example, in the (parent) Red Hat distribution, packages can be added from the (standard) main Red Hat channel 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 the cluster manager. For example, outside the main Red Hat channel, but still within the parent distribution of RHEL7, there is an extra, supplementary, and an optional packages channel. Packages from these channels can be added to the base-distribution to enhance the capabilities of the image or resolve dependencies of the cluster manager. Section 7.4.1 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 the cluster manager, 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 `CENTOS7`, `CENTOS7U9`, `CENTOS8U4`, `CENTOS8U5`, `ROCKY8U4`, `ROCKY8U5`, `RHEL7U9`, `RHEL8U4`, `RHEL8U5`, `SLES12`, `SLES12SP5`, `SLES15`, `SLES15SP2`, `UBUNTU1804`, `UBUNTU2004`.

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 cluster manager repository files, installing distribution package, finalizing image services, and installing the cluster manager 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:

- Using RIMA View via the clickpath `Provisioning`→`Software Images`⇒`Settings`→`Path`
- In `cmsh` within `softwareimage` mode, for example:

```
[rima92->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

7.6.3 Configuring Local Repositories For Linux Distributions, And For The Cluster Manager 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 7.6.2, 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 three 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 cluster manager ISO for CentOS so that a cluster that is completely offline still has a complete and consistent repository access.

Thus, a summary list of what these procedures are about is:

- Setting up a local repository for SLES (page 305)
- Setting up a local repository for RHEL (page 305)
- Setting up a local repository for CentOS and RIMA from the cluster manager ISO for CentOS (page 305)

Configuring Local Repositories For SLES For A Software Image

For SLES11 SP0, SLES11 SP1, and SLES11 SP2, the required packages are spread across two DVDs, and hence two repositories must be created. Assuming the image directory is `/cm/images/sles11sp1-image`, while the names of the DVDs are `SLES-11-SP1-SDK-DVD-x86_64-GM-DVD1.iso` and `SLES-11-SP1-DVD-x86_64-GM-DVD1.iso`, then the contents of the DVDs can be copied as follows:

```
mkdir /mnt1 /mnt2
mkdir /cm/images/sles11sp1-image/root/repo1
mkdir /cm/images/sles11sp1-image/root/repo2
mount -o loop,ro SLES-11-SP1-SDK-DVD-x86_64-GM-DVD1.iso /mnt1
cp -ar /mnt1/* /cm/images/sles11sp1-image/root/repo1/
mount -o loop,ro SLES-11-SP1-DVD-x86_64-GM-DVD1.iso  /mnt2
cp -ar /mnt2/* /cm/images/sles11sp1-image/root/repo2/
```

The two repositories can be added for use by `zypper` in the image, as follows:

```
chroot /cm/images/sles11sp1-image
zypper addrepo /root/repo1 "SLES11SP1-SDK"
zypper addrepo /root/repo2 "SLES11SP1"
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 RIMA 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 `bright-centos.iso`, then its filesystem can be mounted by the root user on a new mount, `/mnt1`, as follows:

```
mkdir /mnt1
mount -o loop bright-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/cm9.2-dvd.repo`

  can be created, for example, for a release tagged with a <*subminor*> number tag, with the content:

  ```
  [rima-repo]
  name=RIMA Cluster Manager DVD Repo
  baseurl=file:///mnt1/data/packages/9.2-<subminor>
  enabled=1
  gpgcheck=1
  exclude = slurm* pbspro* uge* cm-hwloc
  ```

- **for the regular node image:** A repository configuration file

  `$imagedir/etc/yum.repos.d/cm9.2-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@rima92 ~]# yum clean all
[root@rima92 ~]# yum repolist -v
bright-repo                 Bright Cluster 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.

7.6.4 Creating A Custom Image From The Local Repository

After having created the local repositories for SLES, RHEL or CentOS/Rocky (section 7.6.3), 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.

7.7 Creating Images For Other Distributions And Architectures (Multidistro And Multiarch)

RIMA Cluster Manager version 9.0 onward makes it easier to mix distributions in the cluster. This ability is called *multidistro*. However, it is often also loosely called *multiOS*.

RIMA Cluster Manager version 9.0 also introduced the ability to run on certain mixed architecture combinations. The ability to run on multiple architectures is called *multiarch*. For the cluster manager, 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 RIMA Cluster Manager 9.2 are shown in the following table 7.1:

		Head					
		RHEL7	RHEL8	RHEL9	Ubuntu 18.04, 20.04, 22.04	SLES12	SLES15
		(x86-64)	(x86-64, aarch64)	(x86-64, aarch64)	(x86-64, aarch64)	(x86-64)	(x86-64)
Image	RHEL7	x86-64	x86-64	x86-64	x86-64	x86-64	x86-64
	RHEL8	x86-64	x86-64, aarch64	x86-64, aarch64	x86-64, aarch64	x86-64	x86-64
	RHEL9	x86-64	x86-64, aarch64	x86-64, aarch64	x86-64, aarch64	x86-64	x86-64
	Ubuntu 18.04	x86-64	x86-64, aarch64	x86-64, aarch64	x86-64, aarch64	x86-64	x86-64
	Ubuntu 20.04	-	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
	SLES12	x86-64	x86-64	x86-64	x86-64	x86-64	x86-64
	SLES15	x86-64	x86-64	x86-64	x86-64	x86-64	x86-64

Table 7.1: *Images generated by* `cm-image` *that work with head nodes, per architecture and distribution*

For example, a head node running Ubuntu 18.04, on x86-64 or ARMv8 hardware, can support an Ubuntu 18.04, Ubuntu 20.04, RHEL8, and RHEL9 distribution running on x86-64 and on ARMv8 hardware for the compute nodes. In addition, that same head node supports running RHEL7, SLES12, and SLES15 on the compute nodes, but only for x86-64 hardware.

To configure multiarch and multidistro, the `cm-image` tool is used.

7.7.1 The `cm-image` Tool

The `cm-image` tool is essentially a wrapper for the `cm-create-image` (section 7.6) tool. The `cm-image` tool however has some extra features, including allowing the cluster administrator

- to bootstrap the creation of an image from an online or offline repository for some distributions [1]
- 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

[1] The `--bootstrap` option can only create Ubuntu 20.04 images.
In addition, the glibc library on the image kernel needs to be sufficiently compatible with the head node kernel. This is not the case, for example, for a head node running RHEL7, if attempting to create an image of Ubuntu 20.04. In this case, the Ubuntu 20.04 standard C library is too new for the RHEL7 kernel, and image creation fails.
Head node distributions that support the bootstrap option are RHEL7/8 and derivatives, Ubuntu 18.04, Ubuntu 20.04, and SLES15.

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 its first use.

The command options of `cm-image` are illustrated by the following modes and options tree:

```
cm-image
|---------- shell
|             -h|--help
|             -i|--image <image>
|
|---------- create
|           |----- all
|           |        -h|--help
|           |        -f|--force
|           |        -z|--custom-pre-install-script <custom pre-install script>
|           |        --source <archive|directory|host>            (mandatory)
|           |        --bootstrap
|           |        --add-only
|           |        -b|--baserepo <base repository>
|           |        -c|--cmrepo <cluster manager repository>
|           |        -x|--excludecm <RIMA packages to exclude from installation>
|           |        -j|--excludedist <distribution packages to exclude from installation>
|           |        --baseurl <base distribution repository URL>
|           |        --extraurl <extra repository URL>
|           |        --air-gapped
|           |
|           |----- node-installer
|           |        -h|--help
|           |        -f|--force
|           |        -z|--custom-pre-install-script <custom pre-install script>
|           |        --source <archive|directory|host>            (mandatory)
|           |        --bootstrap
|           |        --add-only
|           |        --add-archos
|           |        -b|--baserepo <base repository>
|           |        -c|--cmrepo <cluster manager repository>
|           |        -x|--excludecm <RIMA packages to exclude from installation>
|           |        -j|--excludedist <distribution packages to exclude from installation>
|           |        --baseurl <base distribution repository URL>
|           |        --extraurl <extra repository URL>
|           |        --air-gapped
|           |        --default
|           |
|           |----- swimage
|           |        -h|--help
|           |        -f|--force
|           |        -z|--custom-pre-install-script <custom pre-install script>
|           |        --source <archive|directory|host>            (mandatory)
|           |        --bootstrap
|           |        --add-only
|           |        --add-archos
|           |        -b|--baserepo <base repository>
|           |        -c|--cmrepo <cluster manager repository>
|           |        -x|--excludecm <RIMA packages to exclude from installation>
|           |        -j|--excludedist <distribution packages to exclude from installation>
|           |        --baseurl <base distribution repository URL>
```

```
|           |        --extraurl <extra repository URL>
|           |        --air-gapped
|           |
|           |----- fromfile <JSON input file>
|           |        -h|--help
|           |
|           '----- cmshared
|                    -h|--help
|                    -f|--force
|                    -i|--image <image>
|                    -b|--baserepo <base repository>
|                    -c|--cmrepo <cluster manager repository>
|                    -x|--excludecm <RIMA packages to exclude from installation>
|                    -j|--excludedist <distribution packages to exclude from installation>
|                    --baseurl <base distribution repository URL>
|                    --extraurl <extra repository URL>
|           |        --air-gapped
|                    --default
|                    --add-only
|
|---------- remove
|             -a|--arch <architecture>                    (mandatory)
|             -d|--distro <distribution>                  (mandatory)
|             -h|--help
|             -f|--force
|             --erase
|
'---------- package
              -h|--help
              -i|--image <image>
              --install <package to install>
              --remove <package 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:
  - `rhel7u7`, `rhel7u8`, `rhel7u9`
  - `rhel8u0`, `rhel8u1`, `rhel8u2`, `rhel8u3`, `rhel8u4`, `rhel8u5`, `rhel8u6`, `rhel8u7`, `rhel8u8`
  - `rhel9u0`, `rhel9u1`, `rhel9u2`
  - `sles12sp2`, `sles12sp3`, `sles12sp4`, `sles12sp5`
  - `sles15sp1`, `sles15sp2`, `sles15sp3`, `sles15sp4`, `sles15sp5`
  - `ubuntu1804`, `ubuntu2004`, `ubuntu2204`

  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 7.6.1), 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, eg: /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

7.7.2 Multidistro Examples: Provisioning From CentOS 7 Head Node To Ubuntu 18.04 Regular Nodes

Using Bootstrap For Cluster With Network Access To Repositories

```
[root@rima92 ~]# module load cm-image
[root@rima92 ~]# cm-image create all -a x86_64 -d ubuntu1804 --bootstrap
Creating software image default-image-ubuntu1804-x86_64
...
Creating cmshared directory /cm/shared-ubuntu1804-x86_64
running command: 'mount --bind /cm/shared-ubuntu1804-x86_64
/cm/images/default-image-ubuntu1804-x86_64/cm/shared'
running command: 'cm-create-image --image-type cmshared -n default-image-ubuntu1804-x86_64 -d
/cm/images/default-image-ubuntu1804-x86_64 -u -s '
...
Installing CM packages.......................... [  OK  ]
running command: 'umount /cm/images/default-image-ubuntu1804-x86_64/cm/shared'
Adding cmshared entry /cm/shared-ubuntu1804-x86_64
Cloning fsexports entry
Completed
Added new category: default-ubuntu1804-x86_64
Use this category for adding nodes
Completed
```

As suggested by the output, a new category, default-ubuntu1804-x86_64, appears.
A node can be placed in the new category and restarted:

```
[root@rima92 ~]# cmsh
[rima92]% device use node001
[rima92->device[node001]]% set category default-ubuntu1804-x86_64
[rima92->device*[node001*]]% commit
[rima92->device[node001]]%
Thu Jan 16 14:35:08 2020 [notice] rima92: node001 [   UP   ], restart required (category)
[rima92->device[node001]]% reboot
node001: Reboot in progress ...
```

Using ISO For Cluster Without Network Access

A base tar (section 7.6.1) can be used

```
[root@rima92 ~]# module load cm-image
[root@rima92 ~]# cm-image --verbose create all -a x86_64 -d ubuntu1804 --source \
 /root/basetar/data/UBUNTU1804.tar.gz
Creating software image default-image-ubuntu1804-x86_64
...
```

output follows similar to the preceding case where repository access was possible

Adding Several Update Versions Alongside Each Other

In the following session, a CentOS7u8 x86 archived base tar is being added with `cm-image`. A CentOS7u7 software image is then being added with `cm-create-image`, using the same node-installer and `/cm/shared/` images:

```
[root@rima92 ~]# module load cm-image
[root@rima92 ~]# cm-image --verbose create all -a x86_64 -d centos7u8 --source \
 /root/basetar/data/CENTOS7u8.tar.gz
[root@rima92 ~]# cm-create-image -a /run/CENTOS7u7.tar.gz -f -n \
default-image-centos7u7-x86_64 -i /cm/images/default-image-centos7u7-x86_64 -g public
[root@rima92 ~]# cmsh
[rima92]% device use node001
[rima92->device[node001]]% set category default-centos7u8-x86_64
[rima92->device[node001]]% set softwareimage default-centos7u7-x86_64
```

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-ubuntu1804-x86_6`, has been copied over from another cluster:

```
[root@rima92 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-ubuntu1804-x86_64
```

then the local CMDaemon can be configured for it with the `--add-only` option:

```
[root@rima92 ~]# module load cm-image
[root@rima92 ~]# cm-image create swimage -a x86_64 -d ubuntu1804 --add-only
```

7.7.3 Multiarch Example: Creating An Image From A Centos 8 Head Node For ARMv8 Architecture Regular Nodes

This section explains how to configure regular node ARMv8 hardware, assuming the cluster manager 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@rima92 ~]# mkdir /var/www/html/aarch64-iso
[root@rima92 ~]# mount -o loop /root/bright91-rhel8u2.aarch64.iso /var/www/html/aarch64-iso
```

A repository file can be created with the following content:

```
[root@rima92 ~]# 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 9.2 packages are in the 9.2 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@rima92 ~]# module load cm-image
[root@rima92 ~]# 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@rima92 ~]# cmsh
[rima92]% category list
Name (key)               Software image                        Nodes
------------------------ ------------------------------------- --------
default-centos8-x86_64   default-image                         1
default-rhel8-aarch64    default-image-rhel8-aarch64           1
[rima92]% softwareimage list
Name (key)                  Path                                  ...
--------------------------- ------------------------------------- ...
default-image               /cm/images/default-image              ...
default-image-rhel8-aarch64 /cm/images/default-image-rhel8-aarch64 ...
[rima92]% 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@rima92 ~]# cmsh
[rima92]% device use arm-node001
[rima92->device[node001]]% set category default-rhel8-aarch64
[rima92->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

```
[rima92->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....
[rima92->partition[base]->archos]% use aarch64/ubuntu1804
[rima92->partition[base]->archos[aarch64/ubuntu1804]]% set primaryimage
default-image       default-image-ubuntu1804-aarch64  new-image-ubuntu1804-aarch64
[rima92->...->archos[aarch64/ubuntu1804]]% set primaryimage new-image-ubuntu1804-aarch64
[rima92->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@rima92 ~]# cmsh
[rima92]% category use default-rhel8-aarch64
[rima92->category[default-rhel8-aarch64]]% set bootloaderprotocol tftp
[rima92->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@rima92 ~]# cmsh
[rima92]% softwareimage use default-rhel8-aarch64
[rima92->softwareimage[default-rhel8-aarch64]]% set kerneloutputconsole ttyAMA0
[rima92->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@rima92 ~]# cmsh
[rima92]% device use arm-node001
[rima92->device[node001]]% bmcsettings; set extraarguments "-L USER -t 0x30"
[rima92->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@rima92 ~]# cmsh
[rima92]% device monitoringresources arm-node001 | grep a64fx
a64fx
```

The monitoring settings for IPMI via the `a64fx` object can be enabled as follows:

```
[root@rima92 ~]# cmsh
[rima92]% monitoring setup; use ipmi
[rima92->monitoring->setup[ipmi]]% set script /cm/local/apps/cmd/scripts/metrics/sample_ipmitool.py
[rima92->monitoring->setup*[ipmi*]]% executionmultiplexers
[rima92->monitoring->setup*[ipmi*]->executionmultiplexers]% remove ipmi
[rima92->monitoring->setup*[ipmi*]->executionmultiplexers*]% add resource a64fx
[rima92->monitoring->setup*[ipmi*]->executionmultiplexers*[a64fx*]]% set resources a64fx
[rima92->monitoring->setup*[ipmi*]->executionmultiplexers*[a64fx*]]% commit
```

8

Monitoring: Monitoring Cluster Devices

The cluster manager 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 RIMA 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 cluster manager monitoring system are then described and discussed in further depth. These include visualization of data, concepts, configuration, monitoring customization and `cmsh` use.

8.1 A Basic Monitoring Example And Action

8.1.1 Synopsis Of Basic Monitoring Example

In section 8.1, after an overview (section 8.1.1), a minimal basic example of monitoring a process is set up (section 8.1.2) and used (section 8.1.3). 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

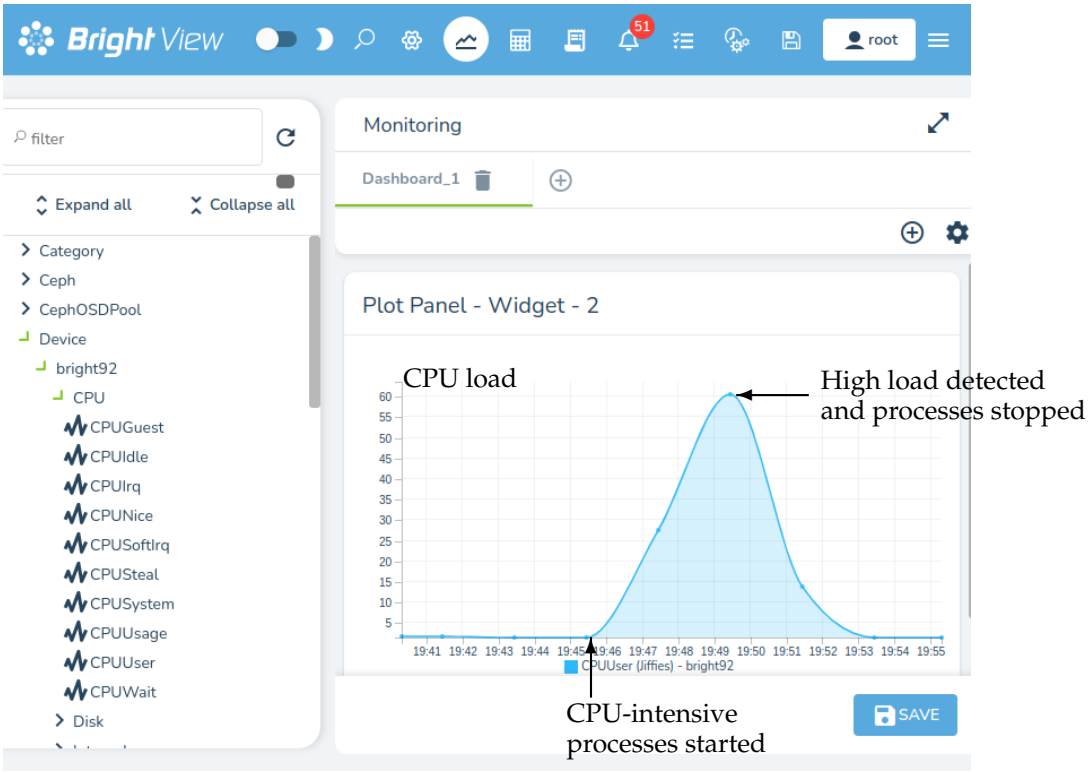


Figure 8.1: Monitoring Basic Example: CPU-intensive Processes Started, Detected And Stopped

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 8.1).

The basic example thus illustrates how the cluster manager monitoring can be used to detect something on the cluster and how an action can be set up and triggered based on that detection.

8.1.2 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 `%user`, 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.

8.1.3 Using The Basic Monitoring Example

Now that the pieces are in place, the administrator can use RIMA 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 RIMA View:

- The clickpath

  `Monitoring→Actions→Monitoring Actions→killprocess→Clone x 1`

  is used to clone the structure of an existing action. The `killprocess` action is convenient because it is expected to function in a similar way, so its options 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 316).

After saving, the `killallyesaction` action becomes part of the list of monitoring actions (figure 8.2).

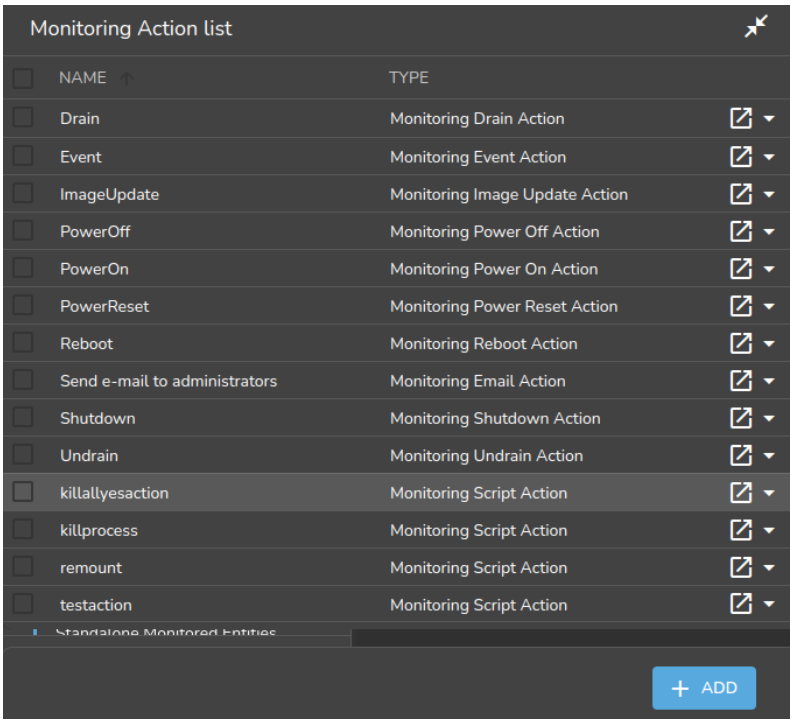


Figure 8.2: RIMA View Monitoring Configuration: Adding An Action

Setting Up A Trigger Using CPUUser On The Head Node(s)

The clickpath

`Monitoring→Triggers→Failing health checks→Clone x 1`

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 8.2.3) `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 8.3):

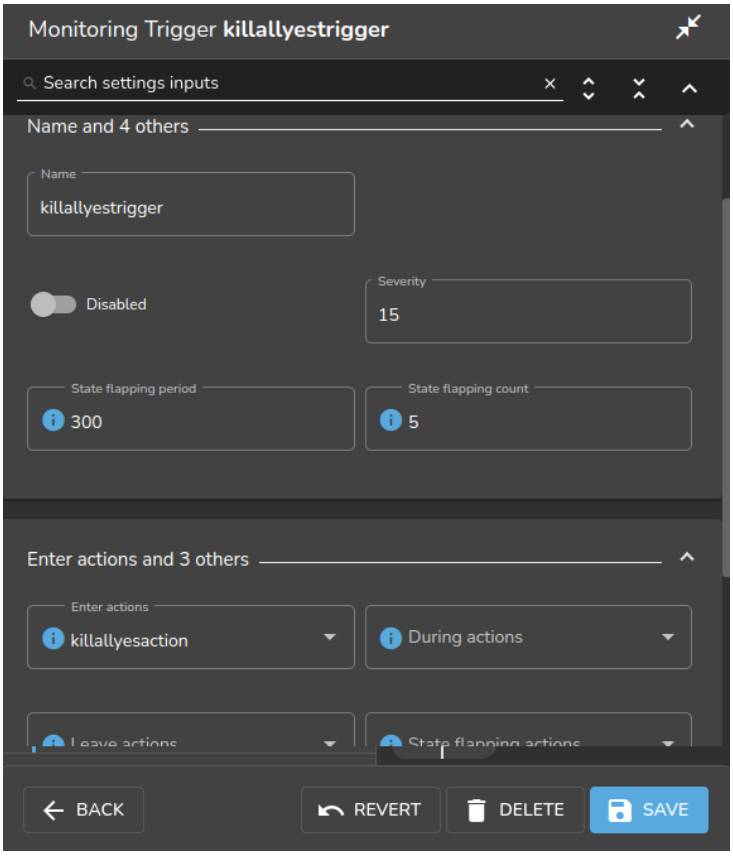


Figure 8.3: RIMA View Monitoring Configuration: Setting A Trigger

- A name is set for the trigger. The name can be arbitrary, and `killallyestrigger` is used in this example.
- The trigger is enabled.
- The `Enter actions` field is filled with the `killallyesaction`, which is the action defined earlier.
- The trigger is saved by clicking on the `SAVE` 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 the `JUMP TO→Expression` button. The button is found in the screen of figure 8.3 by scrolling to the top. Clicking the button brings up the `Monitoring Expression` subwindow (figure 8.4):

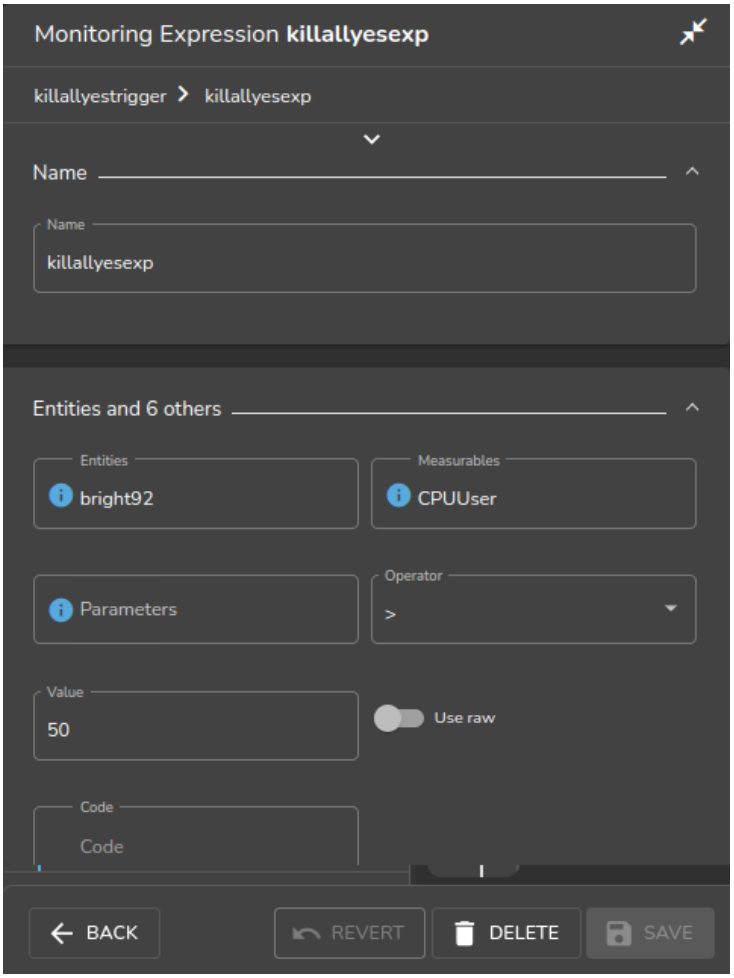


Figure 8.4: RIMA View Monitoring Configuration: Setting An Expression

Within the expression 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 `rima92` in this example, then `rima92` 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 RIMA View, as seen in figure 8.10). 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 8.1, 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 the cluster manager monitoring in greater detail.

8.2 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 the cluster manager covered later on in this chapter will then be understood more clearly.

8.2.1 Measurables

Measurables are measurements (sample values) that are obtained via data producers (section 8.2.10) 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 8.2.2.
- *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 8.2.3.
- *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 8.2.4.

`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 the cluster manager 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 the cluster manager are the following:

`device category partition[base] softwareimages`

However, more generally, an entity can be an object from the following top level modes of `cmsh`:

`category ceph cloud cmjob 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@rima92 ~]# 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 cluster manager cluster can be very diverse. This makes entities a powerful and versatile concept in the cluster manager'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

```
[rima92->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 8.2.2), `metrics` (section 8.2.3), or `healthchecks` (section 8.2.4) command.

In RIMA View, all the entities that are using health checks can be viewed via the clickpath:

`Monitoring→All Health Checks` (figure 8.22 section 8.4.7)

Listing Entities That Use A Measurable

The entities using a specific measurable can be listed with the `usage` command:

Example

```
[rima92->monitoring->measurable]% usage nfs_v3_server_total
Measurable           Count  Entities
-------------------- ------ --------------
nfs_v3_server_total  1      rima92
```

If the number of measurables is too large to view on the screen:

Example

```
[rima92->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

```
rima92->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,
                     rima92
```

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

```
[rima92->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 8.2.2), `list metric` (section 8.2.3), or `list healthcheck` (section 8.2.4).

In RIMA View, the equivalent to listing the measurables can be carried out via the clickpath:

`Monitoring`→`Measurables` (figure 8.11, section 8.4.2)

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

```
[rima92->monitoring->measurable]% use devicestatus
[rima92->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
```

8.2.2 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 2022), `DeviceStatus` is the only enummetric. This may change in future versions of the cluster manager.

The full list of possible values for the enummetric `DeviceStatus` is:

`up`, `down`, `closed`, `installing`, `installer_failed`, `installer_rebooting`, `installer_callinginit`, `installer_unreachable`, `installer_burning`, `burning`, `unknown`, `opening`, `going_down`, `pending`, and `no data`.

The enummetrics available for use can be listed from within the `measurable` submode of the `monitoring` mode:

Example

```
[rima92->monitoring->measurable]% list enum
Type   Name (key)              Parameter           Class     Producer
------ ----------------------- ------------------- --------- -------------------
Enum   DeviceStatus                                Internal  DeviceState
[rima92->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

```
[rima92->device]% enummetrics node001
Type   Name                   Parameter           Class     Producer
------ ----------------------- ------------------- --------- -------------------
Enum   DeviceStatus                                Internal  DeviceState
[rima92->device]%
```

The states that the entity has been through can be viewed with a `dumpmonitoringdata` command (section 8.6.4):

Example

```
[rima92->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 323).

8.2.3 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 8.1, 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 the cluster manager 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

```
[rima92->monitoring]% measurable list metric
Type    Name (key)               Parameter      Class              Producer
------- ------------------------ -------------- ------------------ -------------
Metric  AlertLevel               count          Internal           AlertLevel
Metric  AlertLevel               maximum        Internal           AlertLevel
...
```

In RIMA View, the metrics can be viewed with the clickpath:

`Monitoring`→`Measurables` (figure 8.11, section 8.4.2)

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

```
[rima92->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

```
[rima92->monitoring->measurable]% use alertlevel:count
[rima92->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 RIMA View clickpath to edit the parameters is:

`Monitoring→Measurables→Edit`

8.2.4 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 RIMA View, the health checks that can be configured for all entities can be seen with the clickpath:

`Monitoring`→`Measurables` (figure 8.11, section 8.4.2)

Options can be set for each health check by clicking through via the `Edit` button.

All Configured Health Checks

In RIMA View, health checks that have been configured for all entities can be seen with the clickpath:

`Monitoring`→`All Health Checks` (section 8.4.7)

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 clickpath:

`Monitoring`→`Health status`→<*entity*>→`Show`

Severity Levels For Health Checks, And Overriding Them

A health check has a settable severity (section 8.2.7) 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 8.2.8) 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 8.1 this is how, when the metric measurable `CPUUser` crosses 50 jiffies/s, the `killallyestrigger` is activated, and the `killallyes` action script is run.

8.2.5 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 8.2.7) associated with it. This value is processed for the `AlertLevel` metric (section 8.2.8) when an action is triggered by a threshold event.

Triggers are discussed further in section 8.4.5.

8.2.6 Action

In the basic example of section 8.1, 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 8.2.9).

The actions that can be run are listed from within the `action` submode of the `monitoring` mode.

Example

```
[rima92->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 RIMA View equivalent is accessible via the clickpath:

`Monitoring`→`Actions` (figure 8.17, section 8.4.4)

Configuration of monitoring actions is discussed further in section 8.4.4.

8.2.7 Severity

Severity is a positive integer value that the administrator assigns for a trigger. It takes one of these 6 suggested values:

Value	Name	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 8.2.8). They can also be set by the administrator in the return values of health check scripts (section 8.2.4).

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 8.2.4).

8.2.8 AlertLevel

AlertLevel is a special metric. It is sampled and re-calculated when an event with an associated `Severity` (section 8.2.7) 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.

8.2.9 Flapping

Flapping, or *State Flapping*, is when a measurable trigger that is detecting changes (section 8.4.5) 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 8.1, 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.

8.2.10 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

```
[rima92->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

```
[rima92->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

```
[rima92->monitoring->setup]% list
```

The equivalent in RIMA View is via the clickpath:

```
Monitoring→Data Producers
```

The data producers configured for an entity, such as a head node `rima92`, can be listed with the `monitoringproducers` command:

Example

```
[rima92->device[rima92]]% monitoringproducers
Type               Name               Arguments    Measurables  Node execution filters
------------------ ------------------ ------------ ------------ ----------------------
AlertLevel         AlertLevel                      3 / 231      <0 in submode>
CMDaemonState      CMDaemonState                   1 / 231      <0 in submode>
ClusterTotal       ClusterTotal                    18 / 231     <1 in submode>
Collection         NFS                             32 / 231     <0 in submode>
Collection         sdt                             0 / 231      <0 in submode>
DeviceState        DeviceState                     1 / 231      <1 in submode>
HealthCheckScript  chrootprocess                   1 / 231      <1 in submode>
HealthCheckScript  cmsh                            1 / 231      <1 in submode>
HealthCheckScript  defaultgateway                  1 / 231      <0 in submode>
HealthCheckScript  diskspace                       1 / 231      <0 in submode>
HealthCheckScript  dmesg                           1 / 231      <0 in submode>
HealthCheckScript  exports                         1 / 231      <0 in submode>
HealthCheckScript  failedprejob                    1 / 231      <1 in submode>
HealthCheckScript  hardware-profile                0 / 231      <1 in submode>
HealthCheckScript  ib                              1 / 231      <0 in submode>
HealthCheckScript  interfaces                      1 / 231      <0 in submode>
HealthCheckScript  ldap                            1 / 231      <0 in submode>
HealthCheckScript  lustre                          1 / 231      <0 in submode>
HealthCheckScript  mounts                          1 / 231      <0 in submode>
HealthCheckScript  mysql                           1 / 231      <1 in submode>
HealthCheckScript  ntp                             1 / 231      <0 in submode>
```

```
HealthCheckScript  oomkiller                1 / 231     <0 in submode>
HealthCheckScript  opalinkhealth            1 / 231     <0 in submode>
HealthCheckScript  rogueprocess             1 / 231     <1 in submode>
HealthCheckScript  schedulers               1 / 231     <0 in submode>
HealthCheckScript  smart                    1 / 231     <0 in submode>
HealthCheckScript  ssh2node                 1 / 231     <1 in submode>
Job                JobSampler               0 / 231     <1 in submode>
JobQueue           JobQueueSampler          7 / 231     <1 in submode>
MonitoringSystem   MonitoringSystem         36 / 231    <1 in submode>
ProcMemInfo        ProcMemInfo              10 / 231    <0 in submode>
ProcMount          ProcMounts               2 / 231     <0 in submode>
ProcNetDev         ProcNetDev               18 / 231    <0 in submode>
ProcNetSnmp        ProcNetSnmp              21 / 231    <0 in submode>
ProcPidStat        ProcPidStat              5 / 231     <0 in submode>
ProcStat           ProcStat                 14 / 231    <0 in submode>
ProcVMStat         ProcVMStat               6 / 231     <0 in submode>
Smart              SmartDisk                0 / 231     <0 in submode>
SysBlockStat       SysBlockStat             20 / 231    <0 in submode>
SysInfo            SysInfo                  5 / 231     <0 in submode>
UserCount          UserCount                3 / 231     <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 RIMA View is discussed further in section 8.4.1.

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 RIMA 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 8.8) 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 that who wants to plot/view the data, or else the data cannot be plotted/viewed. [1] 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.

[1] Even more verbosely: 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 the cluster manager is owned by a user.
For all unowned entities, individual access is equivalent to private access.

On a regular cluster manager 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@rima92 ~]# cmsh
[rima92]% monitoring setup
[rima92->monitoring->setup]% use mounts
[rima92->...[mounts]]% set access private
[rima92->...*[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@rima92 ~]# 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@rima92 ~]# cmsh
[rima92]% monitoring measurable
[rima92->monitoring->measurable]% use loadone
[rima92->...[LoadOne]]% get access
Public (SysInfo)
[rima92->...[LoadOne]]% set access private
[rima92->...*[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@rima92 ~]# cmsh -c "monitoring measurable; list -f name:40,access"
name (key)                               access
---------------------------------------- --------------------
```

```
AlertLevel                          Public
AlertLevel                          Public
AlertLevel                          Public
AvgJobDuration                      Public
BlockedProcesses                    Public
...
```

8.2.11 Conceptual Overview: The Main Monitoring Interfaces Of RIMA View

RIMA View, besides having the default settings mode, has some other display modes and logging view modes that can be selected via the 11 icons in the top right corner of the RIMA View standard display (figure 8.5):

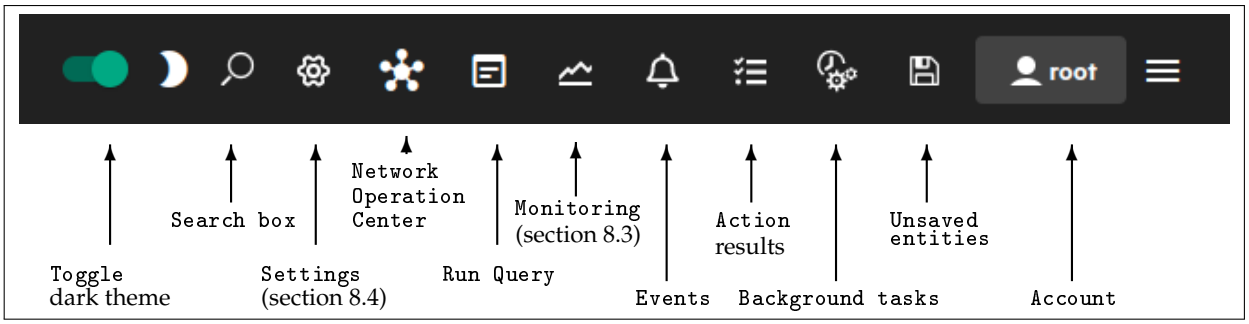


Figure 8.5: RIMA View: Top Right Corner Icons

The 11 icons are described from left to right next:

1. `Toggle dark theme` option allows the display of RIMA View to be toggled to a darker theme.
2. `Search box` allows resource to be searched for, with predictive text suggestions.
3. `Settings` mode is active when RIMA View first starts up.

   The `Settings` mode has a navigation panel to the left of it, showing the resources of the cluster as expandable items. One of the resources is `Monitoring`. This resource should not be confused with the RIMA View `Monitoring` mode, which is launched by the next icon in figure 8.5. The `Monitoring` resource is about configuring how items are monitored and how their data values are collected, and is discussed further in section 8.4.
4. `Network Operation Center` mode allows configured devices to be displayed on a map. Queries can be constructed for running periodically on devices.
5. `Run Query` mode runs a query on a configured device periodically.
6. The `Monitoring` mode allows visualization of the data values collected according to the specifications of the RIMA View `Monitoring` resource. The visualization allows graphs to be configured, and is discussed further in section 8.3.
7. The `Events` icon allows logs of events (section 8.10.1) to be viewed.
8. The `Action results` icon allows the logs of the results of actions to be viewed.
9. The `Background tasks` icon allows background tasks to be viewed.
10. The `Unsaved entities` icon allows entities that have not yet been saved to be viewed.
11. The `Account` handling icon allows account settings to be managed for the RIMA View user.

The monitoring aspects of the first two icons are discussed in greater detail in the sections indicated.

8.3 Monitoring Visualization With RIMA View

The `Monitoring` icon in the menu bar of RIMA View (item 6 in figure 8.5) 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 RIMA 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 8.6). 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 8.5 and 8.6.

Visualization of monitoring graphs with RIMA View is now described.

8.3.1 The Monitoring Window

If the `Monitoring` icon is clicked on from the menu bar of RIMA View (figure 8.5), then a monitoring window for visualizing data opens up. By default, this displays blank plot panels—graph axes with a time scale going back some time on the x-axis, and with no y-axis measurable data values plotted (figure 8.6).

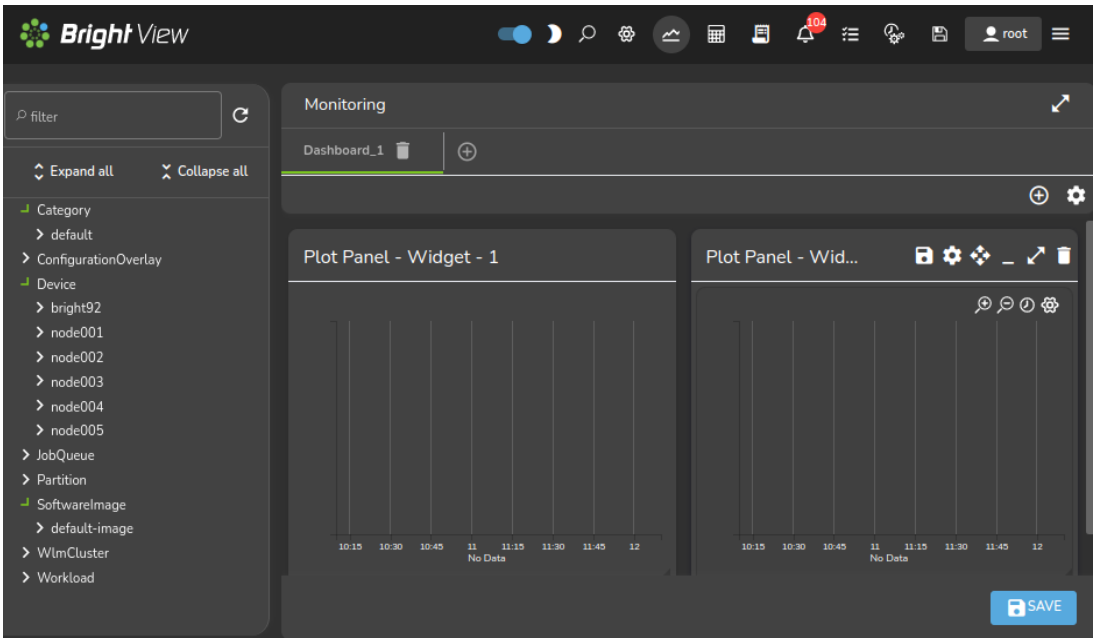


Figure 8.6: RIMA View Monitoring Window: Blank Plot Panels

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 `rima92`, it can be selected from the navigation clickpath:

Menu bar→*Monitoring icon*→`Device`→`rima92`→`CPU`→`CPUUser`

Sometimes, finding a measurable is easier if the `Expand all` widget is used, together with the `Search`

box. Typing in `CPUUser` in the search box then shows all the measurables with that text (figure 8.7). The search is case-insensitive.

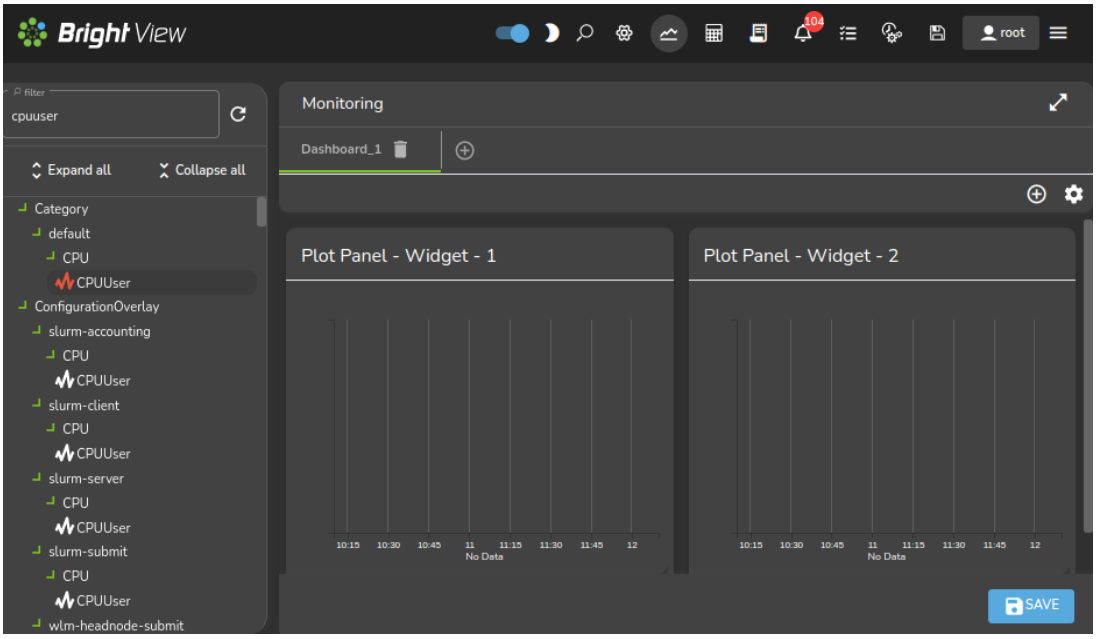


Figure 8.7: RIMA View Monitoring Window: Search Box In Navigation

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.

8.4 Monitoring Configuration With RIMA View

This section is about the configuration of monitoring for measurables, and about setting up trigger actions.

If RIMA View is running in the standard `Settings` mode, which is the gear icon in figure 8.5, page 332, then selecting `Monitoring` from the navigation menu resources makes the following menu items available for managing or viewing:

- `Data Producers` (section 8.4.1)
- `Measurables` (section 8.4.2)
- `Consolidators` (section 8.4.3)
- `Actions` (section 8.4.4)
- `Triggers` (section 8.4.5)
- `Health status` (section 8.4.6)
- `All Health checks` (section 8.4.7)
- `Standalone Monitored Entities` (section 8.4.8)
- `PromQL Queries` (section 8.4.9)
- `Resources` (section 8.4.10)
- `Types` (section 8.4.11)

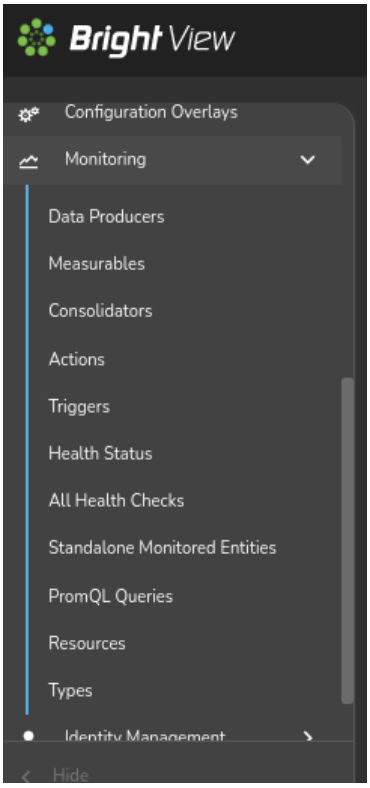


Figure 8.8: RIMA View Monitoring Configuration Settings

These settings (figure 8.8) are now discussed in detail.

8.4.1 Monitoring Configuration: Data Producers

The clickpath:

`Monitoring`→`Dataproducers`

opens up the `Monitoring Data Producer list` screen, which lists all the data producers (figure 8.9).

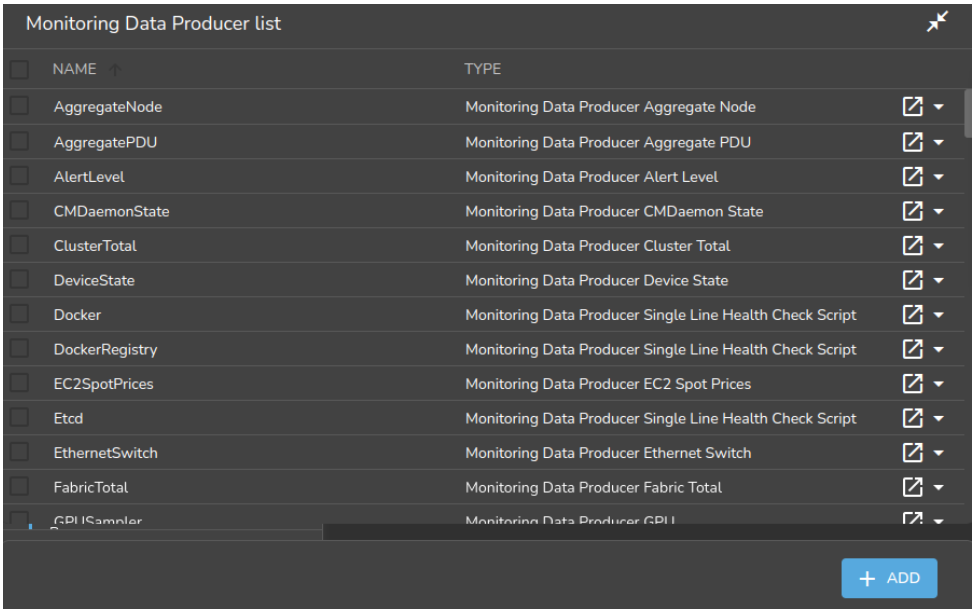


Figure 8.9: RIMA View Monitoring Configuration Data Producers

Data producers are introduced in section 8.2.10.

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 8.10:

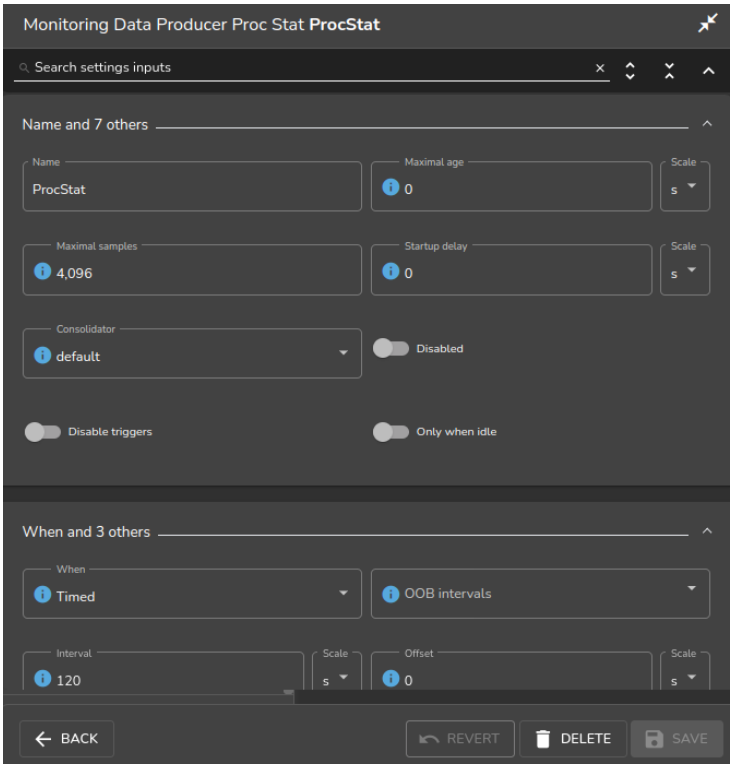


Figure 8.10: RIMA View Monitoring Configuration Data Producer: `ProcStat`

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 the cluster manager 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 8.10 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.
- `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, a day, and a week. Consolidators are explained further in section 8.4.3.

- `Node execution filters`: A way to filter execution (restrict execution) of the data producer. It tells the cluster manager where the data producer runs. If not set, then the data producer runs on all nodes managed by CMDaemon. Filters can be for nodes, types, overlays, resources, and categories.

- `Execution multiplexer`: A way to multiplex execution (have execution work elsewhere) for a data producer. It tells the cluster manager about the entities that the data producer is sampling for. A data producer gathers data at the nodes defined by the node execution filter, and with multiplex execution the data producer gathers samples from other entities. These entities can be nodes, types, overlays, and resources. The entities from which it can sample are defined into groups called execution multiplexers. Execution multiplexers can thus be node multiplexers, type multiplexers, overlay multiplexers, or resource multiplexers.

- `When`: This has six possible values:

  - `Timed`: Data producer is run at a periodic `Interval`. This is the default.
  - `Pre job`: Data producer is only run before a job, in the prolog of the job. .
  - `On demand`: Data producer is only run on demand, and not at a periodic `Interval`.
  - `Timed and prejob`: Data producer is run at a periodic `Interval` and also run before a job.
  - `Out of band`: Data producer is only run on out of band connections.
  - `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 592).

8.4.2 Monitoring Configuration: Measurables

The `Measurables` window lists the available measurables (figure 8.11):

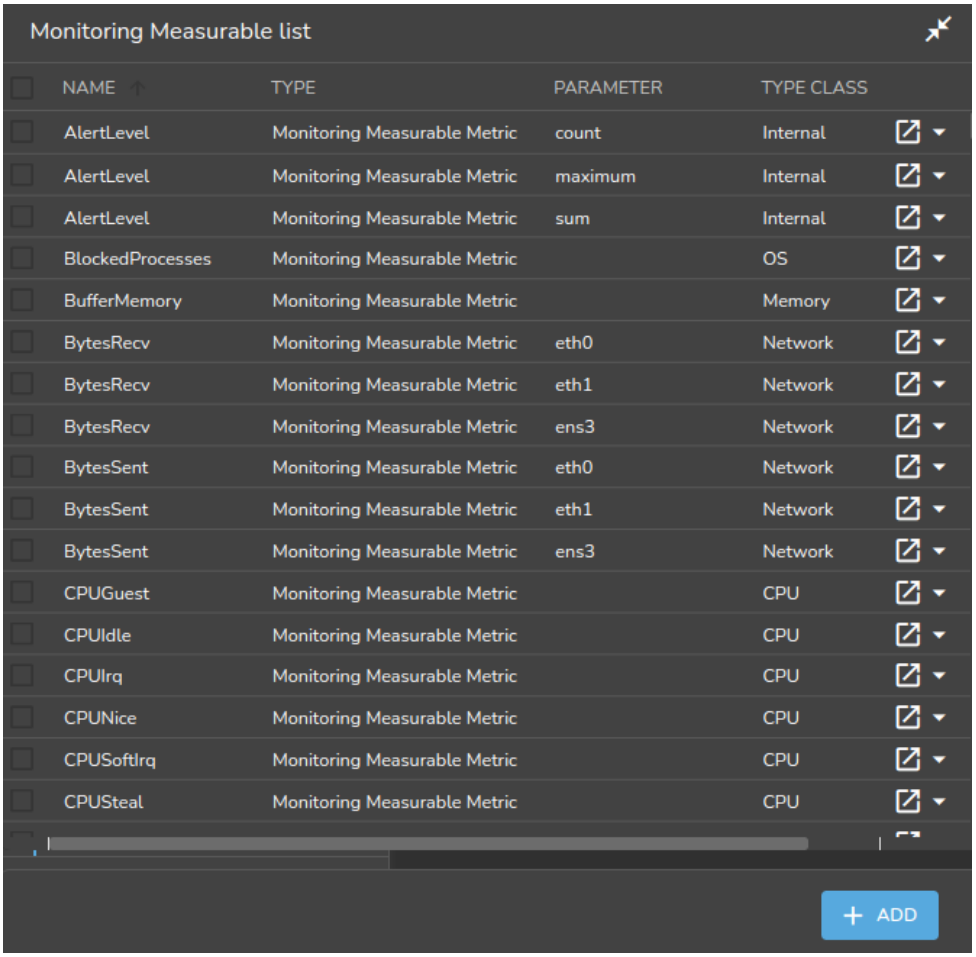


Figure 8.11: RIMA View Monitoring Configuration Measurables

There are many measurables, so using the search box in the menu bar (item 2 in list describing figure 8.5) can be handy.

From the measurables window, a subwindow can be opened with the `Edit` button for a measurable. This accesses the options for a particular measurable (figure 8.12):

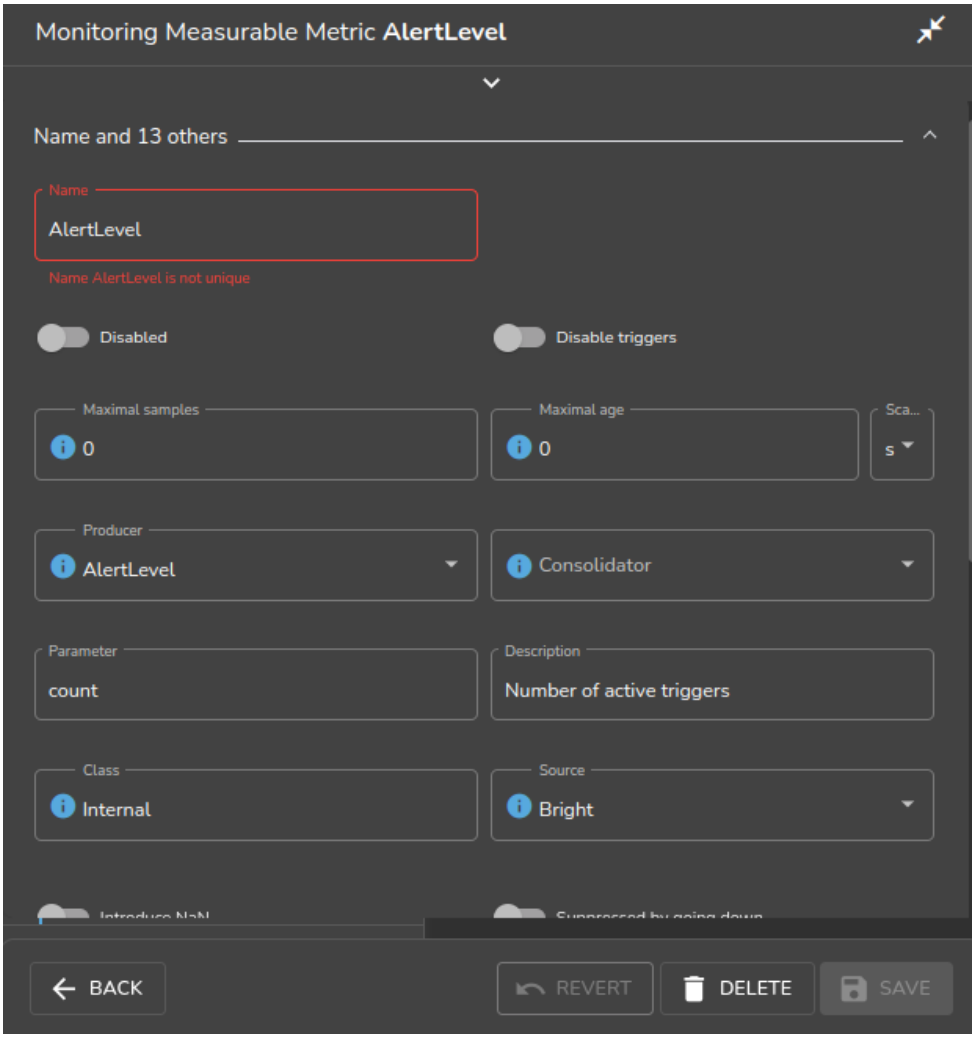


Figure 8.12: RIMA View Monitoring Configuration Measurables Options Subwindow

The options shown include the sampling options: `Maximal age`, `Maximal samples`, and `Consolidator`. The sampling options work as described for data producers (section 8.4.1).

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.

8.4.3 Monitoring Configuration: Consolidators

Introduction To Consolidators

The concept of consolidators is explained using simple ascii graphics in Appendix I, while the `cmsh` interface to the `consolidators` submode is discussed in section 8.5.2.

In this current section, the RIMA View interface to consolidators is discussed.

In RIMA View, the `Monitoring Consolidator list` window lists all consolidator groups (fig-

ure 8.13). There are two pre-existing consolidator groups: `default` and `none`.

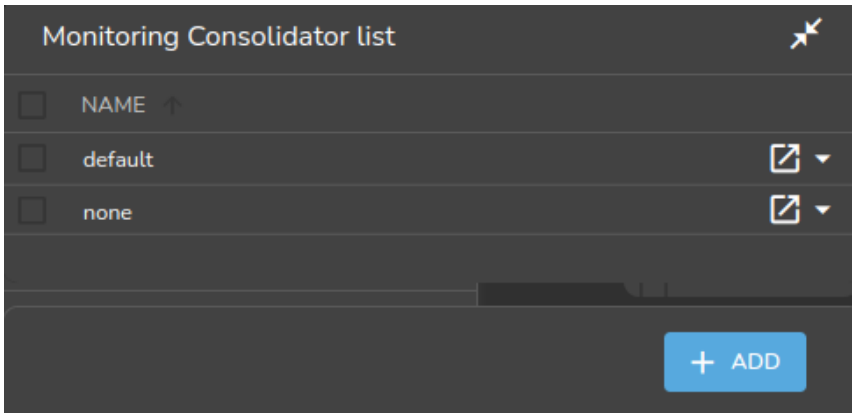


Figure 8.13: RIMA View Monitoring Consolidator list

Subwindows allow the consolidator components (consolidator items) to be created or modified (figure 8.14).

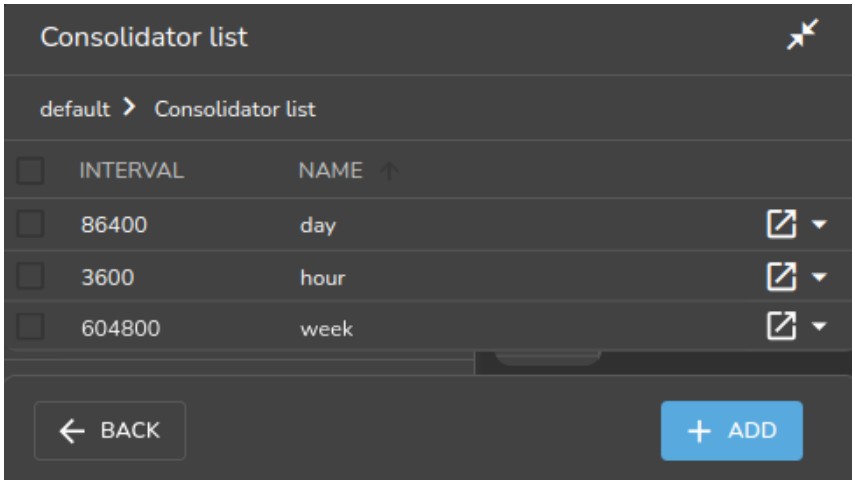


Figure 8.14: RIMA View Monitoring Configuration Consolidator Items

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 8.4.1) for data producers are both set to `0`.

The `default` Consolidator Group

The `default` consolidator group consists of the consolidators `hour`, `day`, and `week`. These 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.

Other Consolidator Group Possibilities

Other sets of custom intervals can also be defined. For example, instead of the `default` consolidator group, consisting of consolidators of an `hour`, a `day`, and a `week`; a similar group called the `decimalminutes` consolidator group, consisting of consolidators of `1min`, `10min`, `100min`, `1000min`, `10000min`, could be created with the appropriate intervals (figure 8.15):

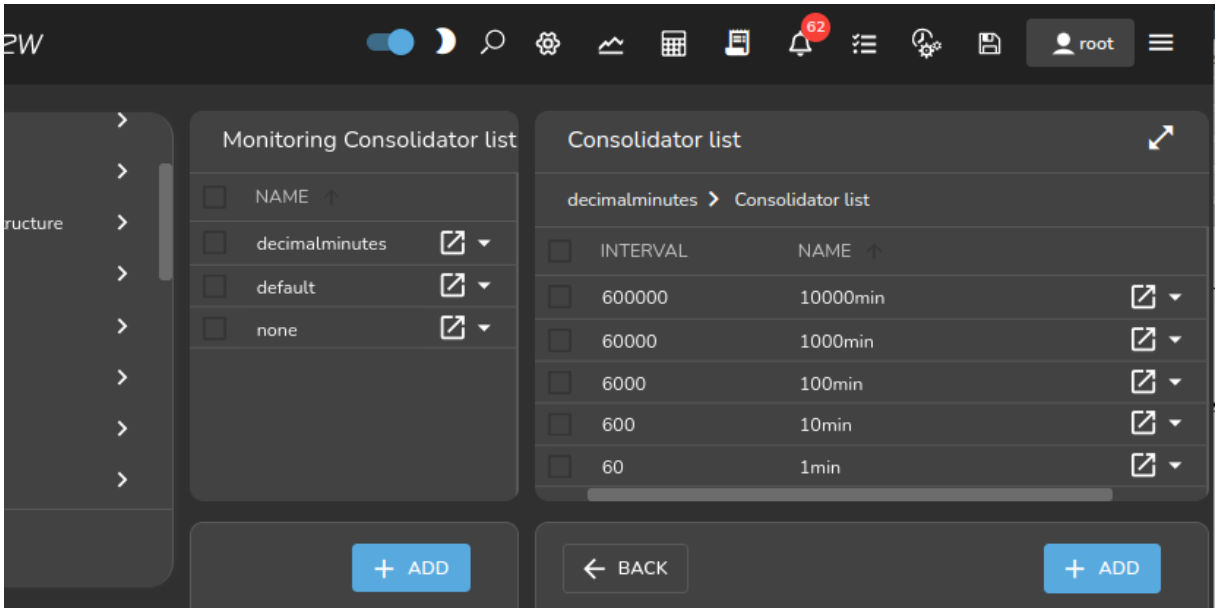


Figure 8.15: RIMA View Monitoring Configuration Consolidators: `decimalminutes` Consolidator Group

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 8.16).

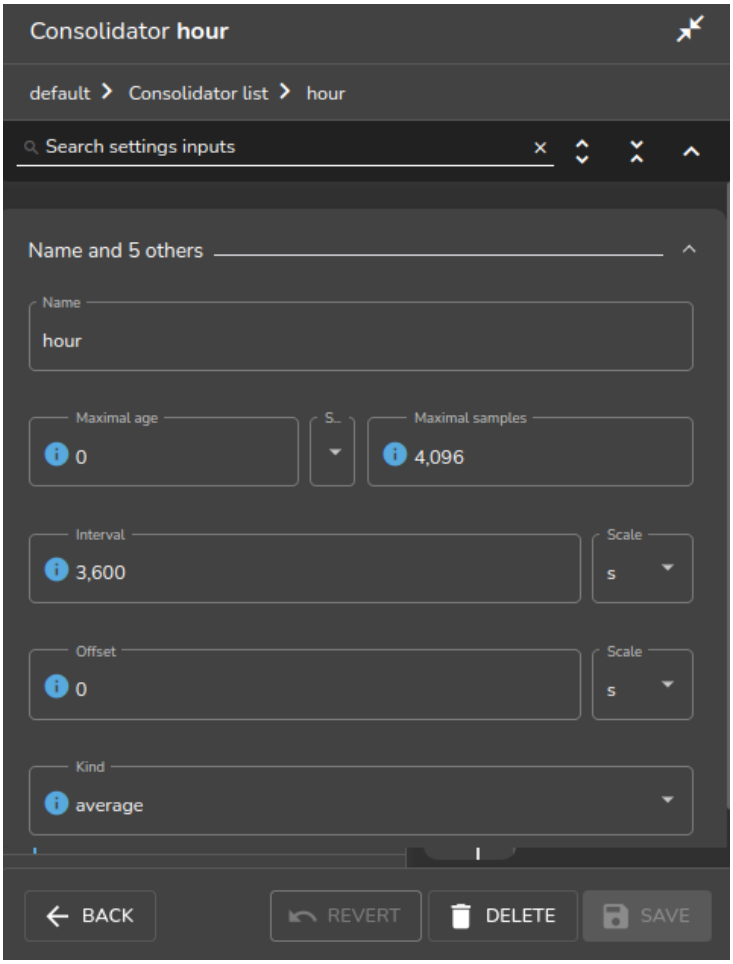


Figure 8.16: RIMA View Monitoring Configuration Consolidators: Consolidator Item Settings

The consolidator item `hour`, which is within the `default` consolidators group, can have its properties edited using the clickpath:

`Monitoring→Consolidators[default]→Edit→Consolidator[hour]→Edit`

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 items with names of `Day`, `Hour`, and `Month` are already set up, with appropriate values for their corresponding fields.
- `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 3600. 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[2].

8.4.4 Monitoring Configuration: Actions

Actions are introduced in section 8.2.6. The Actions window (figure 8.17) displays actions that the cluster manager provides by default, and also displays any custom actions that have been created:

[2]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 $t_{\text{raw gone}}$:

$$t_{\text{consolidation gone}} = (\text{Maximalsamples})_{\text{consolidation}} \times (\text{Interval})_{\text{consolidation}}$$

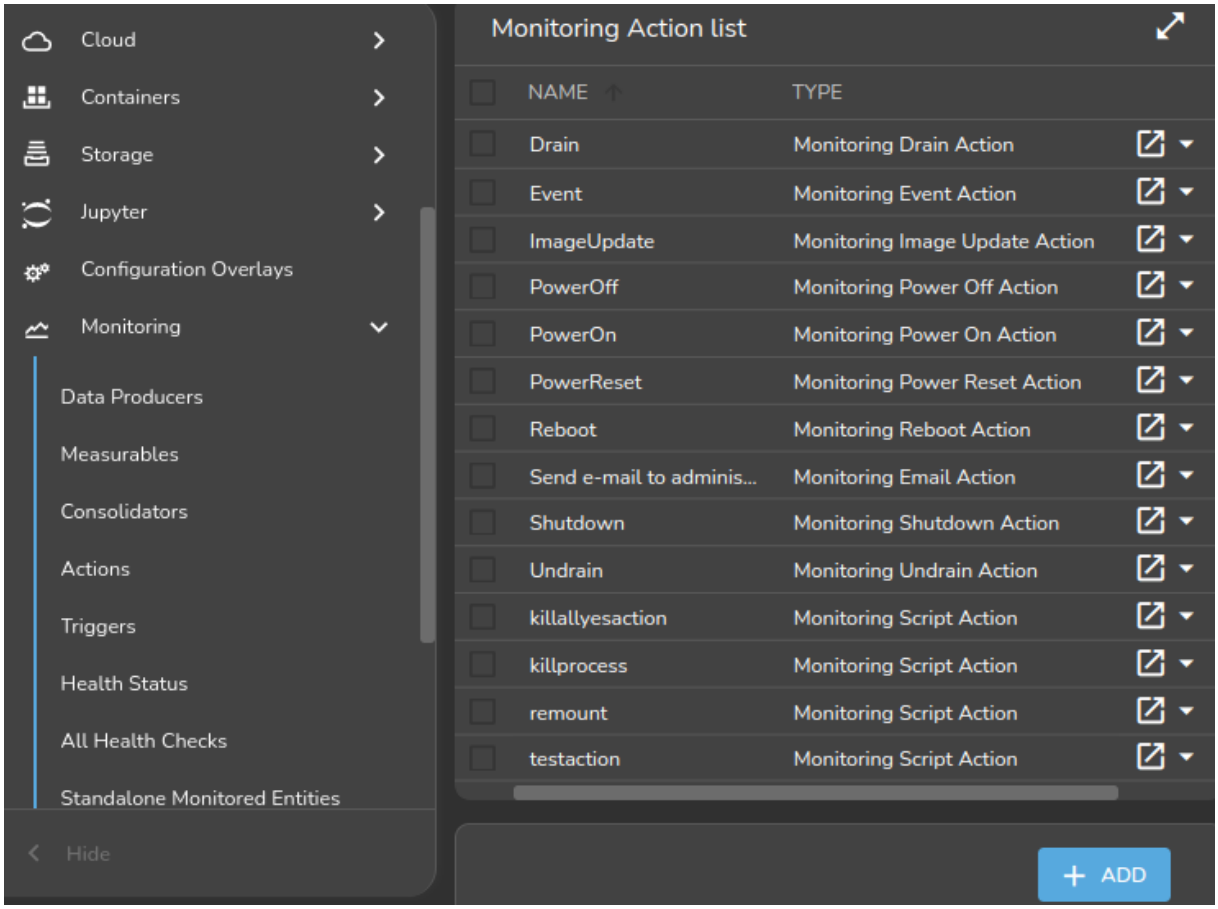


Figure 8.17: RIMA View Monitoring Configuration: Actions

The `killallyes` script from the basic example of section 8.1 would show up here if it has been implemented.

Actions are triggered, by triggers (section 8.4.5).

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
- `Event`: Sends an event to users connected with `cmsh` or RIMA 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 Edit button is clicked. A subwindow with 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 RIMA 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.

  A BNF 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, before the line beginning "Please take action". The default text can be managed 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.

8.4.5 Monitoring Configuration: Triggers

Triggers are introduced in section 8.2.5. The `Triggers` window (figure 8.18) allows actions (section 8.2.6) to be triggered based on conditions defined by the cluster administrator.

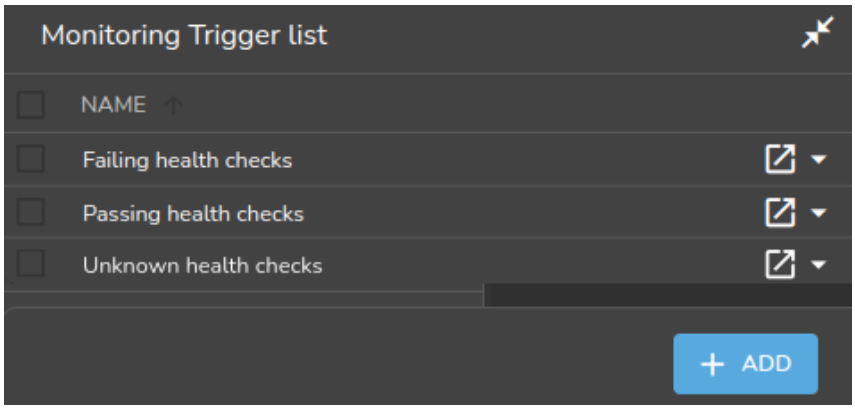


Figure 8.18: RIMA View Monitoring Configuration: Triggers

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: the latest sample value means that either the value has entered a zone when it was not in the zone in the preceding sample, or the latest sample means that the value has left a zone when it was in the zone in the preceding sample
- When the zone remains crossed. That is: the latest sample as well as the preceding sample 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 RIMA 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@rima92 ~]# cmsh
[rima92]% monitoring action
[rima92->monitoring->action]% use send\ e-mail\ to\ administrators
[rima92->...[send e-mail to administrators]]% append recipients user1@example.com
[rima92->...*[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`:

```
[rima92->monitoring->action[use send e-mail to administrators]]% monitoring trigger
[rima92->monitoring->trigger]% use failing\ health\ checks
[rima92->monitoring->trigger[Failing health checks]]% append enteractions send\ e-mail\ to\ administrators
[rima92->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 RIMA View way to carry out the configuration is using the clickpath:

  `Monitoring→Actions→Send e-mail to administrators→Edit`

  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 RIMA View via the clickpath:

  `Monitoring→Triggers→Failing Health Checks→Edit→Enter Actions↓Send E-mail to Administrators`

  The checkbox for the "`Send E-mail to Administrators`" action should be ticked and the configuration saved.

Adding Custom Triggers: Any Measurable, Any Action

More triggers can be added. The `killallyestrigger` example from the basic example of section 8.1, seen in figures 8.3 and 8.4, 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

One of the options presented when editing a trigger listed in figure 8.18 is the `Expression` button. Clicking on it opens up the expression subwindow. The expression for the trigger can then be configured by setting the entity, measurable, parameters, (comparison) operator, and measurable value, as shown in figure 8.19:

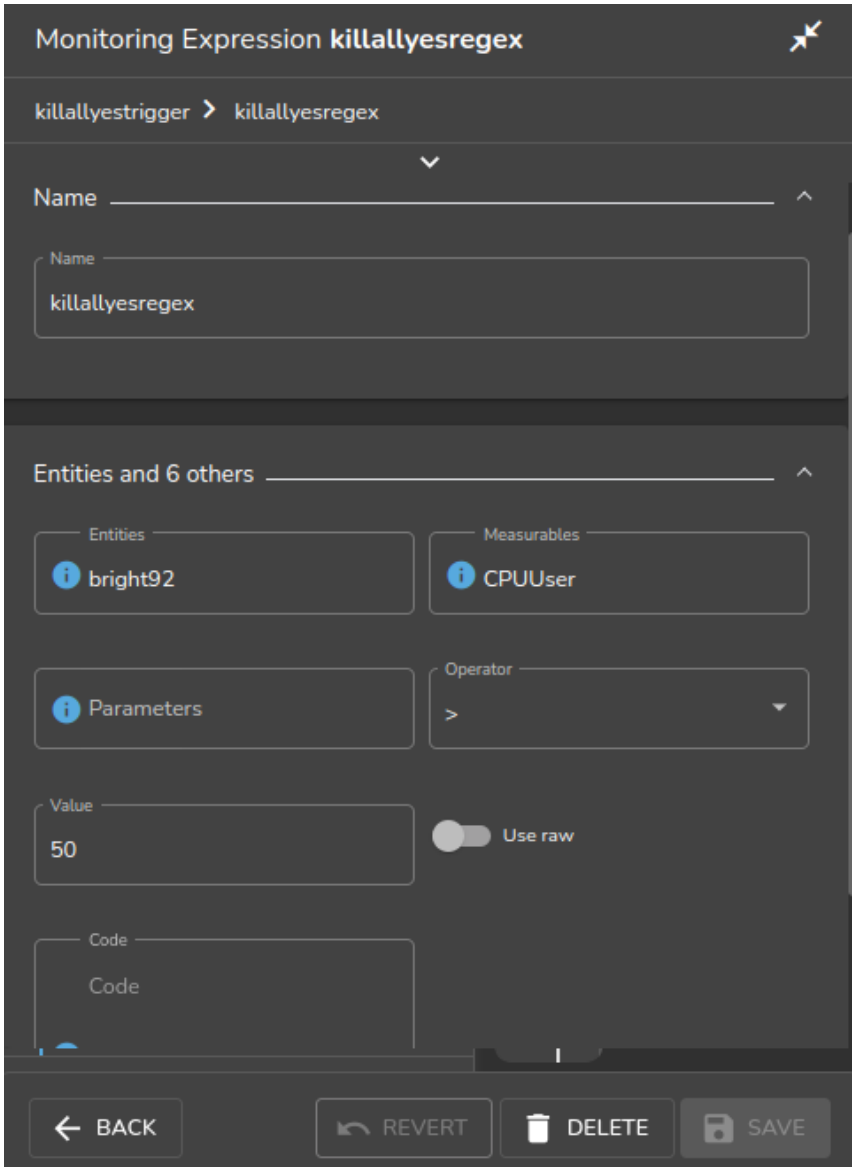


Figure 8.19: RIMA View Monitoring Configuration: Triggers Expression

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@rima92 ~]# cmsh
[rima92]% monitoring trigger add nodedown
[rima92->monitoring->trigger*[nodedown*]]% expression
[rima92->monitoring->trigger*[nodedown*]->expression[compare]]% show
Parameter                      Value
------------------------------ ------------------------------------------------
Name                           compare
Revision
Type                           MonitoringCompareExpression
```

```
Entities
Measurables
Parameters
Operator                         ==
Value                            FAIL
Use raw                          no
[rima92->monitoring->trigger*[nodedown*]->expression*[compare*]]% set value down
[rima92->monitoring->trigger*[nodedown*]->expression*[compare*]]% set operator eq
[rima92->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 37) gives a helpful warning here, so that the cluster administrator can fix the setting:

```
[rima92->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
[rima92->monitoring->trigger*[nodedown*]->expression*[compare*]]% set measurables devicestatus
[rima92->monitoring->trigger*[nodedown*]->expression*[compare*]]% ..;..
[rima92->monitoring->trigger*[nodedown*]]% set enteractions send e-mail to administrators
[rima92->monitoring->trigger*[nodedown*]]% commit
```

8.4.6 Monitoring Configuration: Health status

The `Health status` window (figure 8.20) 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 8.2.7):

Monitoring Health Overview list

ENTITY	MAXIMAL SEVERITY OF ALL FAILED TRIGGERS	TOTAL SEVERITY OF ALL FAILED TRIGGERS	TOTAL COUNT OF ALL FAILED TRIGGERS
bright92	0	0	0
mon001	0	0	0
node001	0	0	0
node002	15	15	1

Figure 8.20: RIMA View Monitoring Configuration: Health Status

In the example shown in figure 8.20 the last entity shows a severity issue, while the other devices are fine. Details of the individual health checks per entity can be viewed in a subwindow using the `Show` button for that entity. Clicking on the show button for the last entity in this example opens up a subwindow (figure 8.21). For this example the issue turns out to be due to a `FAIL` status in the `ssh2node` measurable.

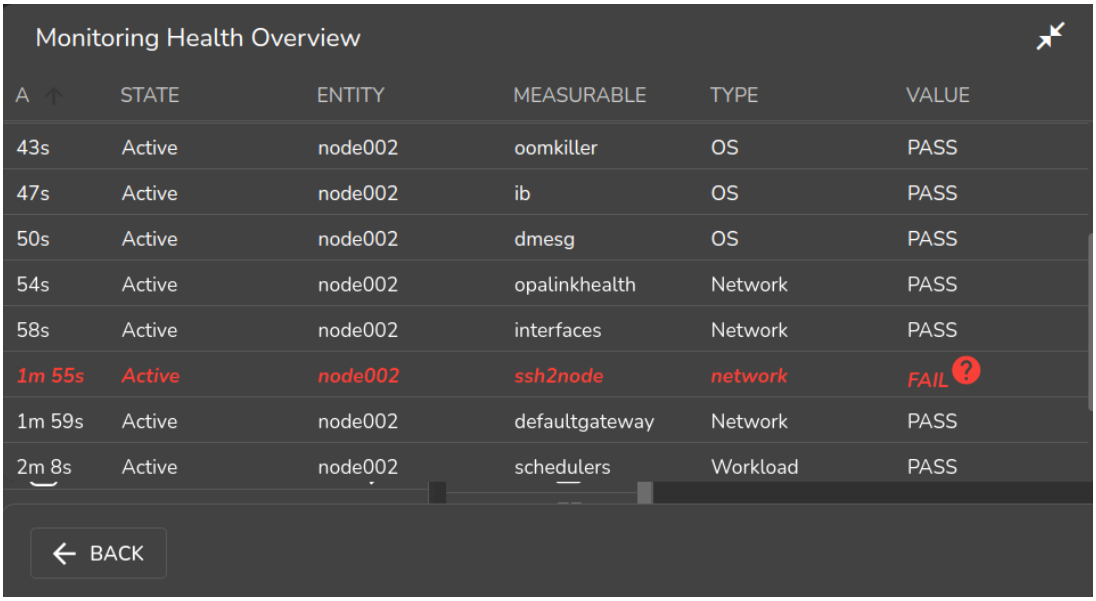


Figure 8.21: RIMA View Monitoring Configuration: Health Status For An Entity

8.4.7 Monitoring Configuration: All Health Checks

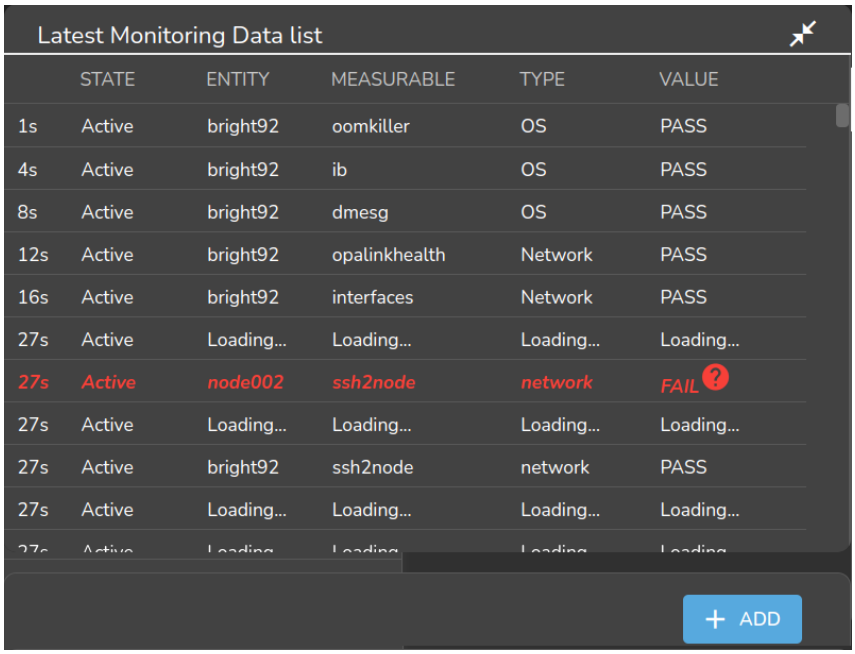


Figure 8.22: RIMA View Monitoring Configuration: All Health Checks For All Entities

The `All Health checks` window shows all the running health checks for all entities. The `Group by Entity` option at the top of the `ENTITY` column can be used to show the results for per entity only. The results for one entityare then similar to what the `Show` button for the entity produces in section 8.4.6, figure 8.21.

8.4.8 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 the cluster manager—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 RIMA View browser could be the standalone entity. This could have its connectivity monitored by pinging it from the head node with a custom script.

8.4.9 Monitoring Configuration: PromQL Queries

The `Prometheus Query list` window displays the list of PromQL job-related queries, and allows query properties to be edited. Drilldowns can also be viewed.

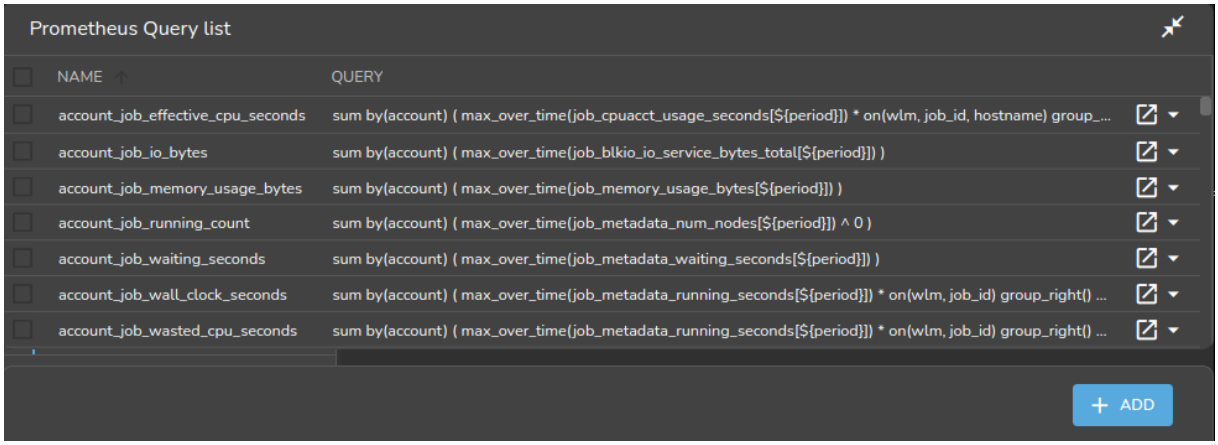


Figure 8.23: RIMA View Monitoring Configuration: PrompQL Queries

PromQL job queries are discussed further in section **??**.

8.4.10 Monitoring Configuration: Resources

The Monitoring Resource list window displays a view-only list of resources.

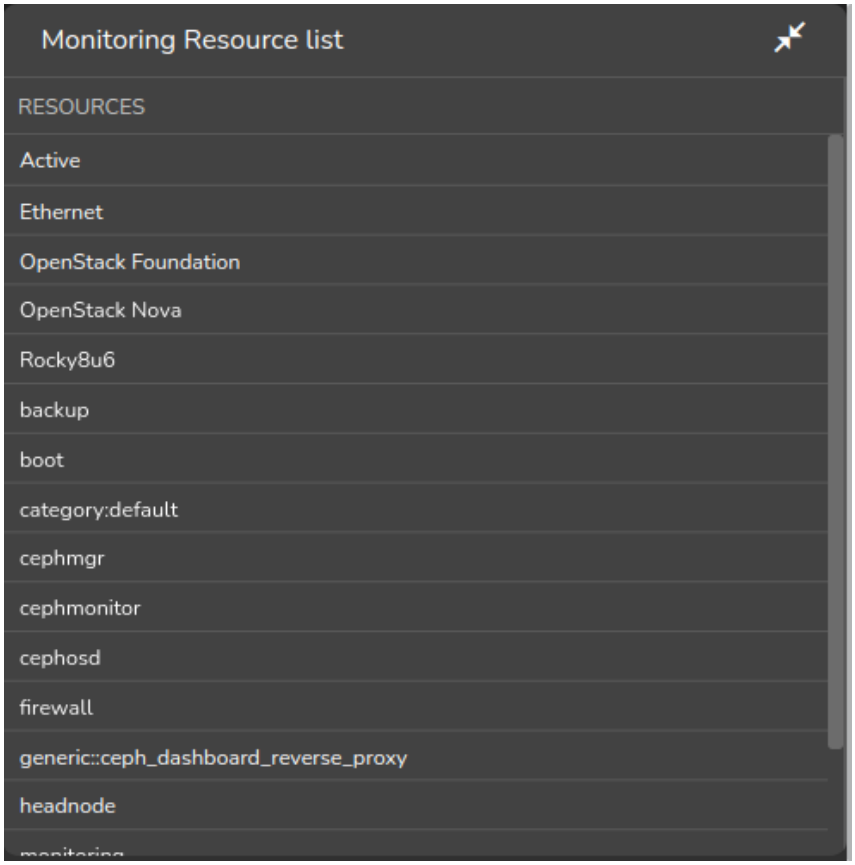


Figure 8.24: RIMA View Monitoring Configuration: Resources

8.4.11 Monitoring Configuration: Types

The Monitoring Types list window displays a view-only list of types.

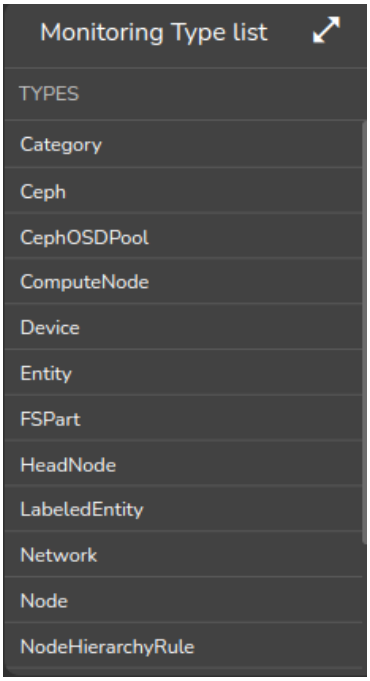


Figure 8.25: RIMA View Monitoring Configuration: Types

8.5 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 RIMA View in section 8.4. Similarly to how monitoring sub-windows are accessed in RIMA 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 -9
============================== 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 8.26.

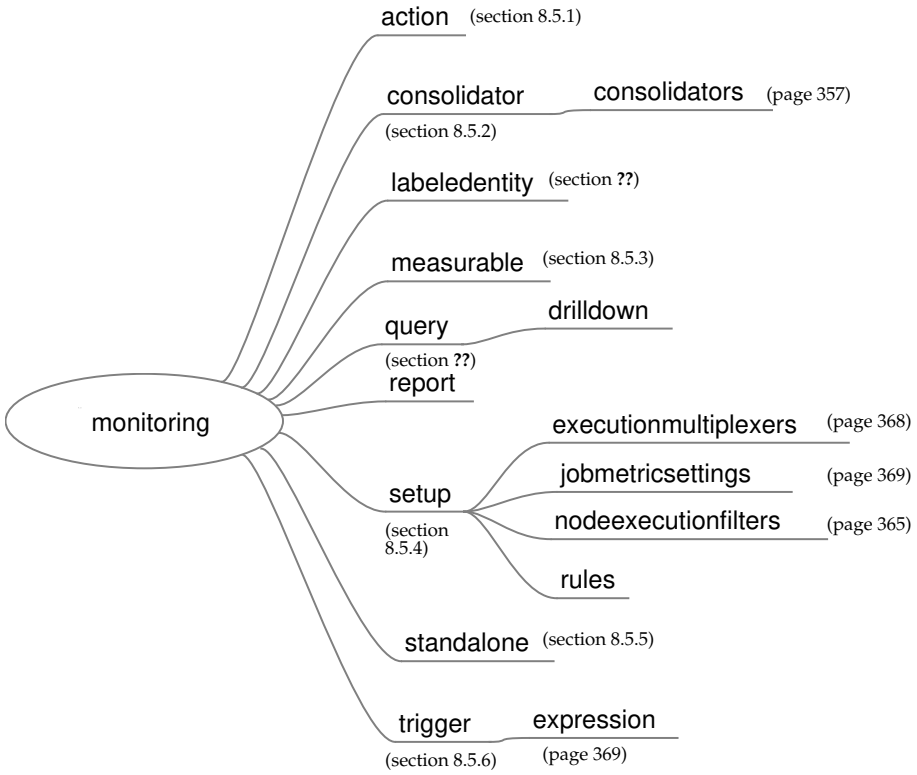


Figure 8.26: Submodes Under monitoring Mode

Sections 8.5.1–8.5.6 give examples of how objects are handled under these monitoring modes. To avoid repeating similar descriptions, section 8.5.1 is relatively detailed, and is often referred to by the other sections.

8.5.1 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 RIMA View clickpath:

```
Monitoring→Actions
```

described earlier in section 8.4.4:

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 give.

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   Drain node from all WLM
Email       Send e-mail to   Active   Send e-mail
             administrators
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      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 (node accepts new WLM jobs)
```

The preceding shows the actions available on a newly installed system.

The `show` command of `cmsh` displays the individual parameters and values of a specified action:

Example

```
[myheadnode->monitoring->action]% show poweroff
Parameter                      Value
------------------------------ --------------------------
Action                         Power off a device
Allowed time
Disable                        no
Name                           PowerOff
Revision
Run on                         Active
Type                           PowerOff
```

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 `get` command returns the value of an individual parameter of the action object:

Example

```
[myheadnode->monitoring->action]% get poweroff runon
active
```

The `action` Submode: `add`, `use`, `remove`, `commit`, `refresh`, `modified`, `set`, `clear`, And `validate`

In the basic example of section 8.1, in section 8.1.2, the `killallyes` action was cloned from a similar script using a `clone` option in RIMA 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 8.1.2 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, 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 8.1, in section 8.1.2. 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.

8.5.2 The consolidator Submode

Consolidators are introduced in section 8.4.3. Consolidators can be managed in cmsh via the consolidator mode, which is the equivalent of the consolidators window (section 8.4.3) in RIMA 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 8.4.3:

```
[rima92->monitoring->consolidator]% list
Name (key)               Consolidators
------------------------ ------------------------
default                  hour, day, week
none                     <0 in submode>
```

Each consolidators entry can have its parameters accessed and adjusted.

For example, the parameters can be viewed with:

Example

```
[rima92->monitoring->consolidator]% use default
[rima92->monitoring->consolidator[default]]% show
Parameter                      Value
------------------------------ ------------------------------------------------
Consolidators                  hour, day, week
Name                           default
Revision
[rima92->monitoring->consolidator[default]]% consolidators
[rima92->monitoring->consolidator[default]->consolidators]% list
Name (key)               Interval
------------------------ ------------------------
day                      1d
hour                     1h
week                     1w
[rima92->monitoring->consolidator[default]->consolidators]% use day
[rima92->monitoring->consolidator[default]->consolidators[day]]% show
Parameter                      Value
------------------------------ ------------------------------------------------
Interval                       1d
Kind                           AVERAGE
Maximal age                    0s
Maximal samples                4096
Name                           day
Offset                         0s
Revision
[rima92->monitoring->consolidator[default]->consolidators[day]]%
```

For the day consolidator shown in the preceding example, the number of samples saved per day can be doubled with:

Example

```
[rima92->monitoring->consolidator[default]->consolidators[day]]% set maximalsamples 8192
[rima92->monitoring->consolidator*[default*]->consolidators*[day*]]% commit
```

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 RIMA View way, where a decimalminutes group is created, is discussed in the example in section 8.4.3, page 342.

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

```
[rima92]% monitoring consolidator
[rima92->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, using the kind setting. Another setting is interval, which defines the interval with which the old data is compressed:

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
------------------- -----------
Interval            1d
Kind                maximum
Maximal age         0s
Maximal samples     4096
Name                max-per-day
Offset              0s
Revision
[...[max-per-day*]]% commit
```

8.5.3 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 RIMA View clickpath:

```
Monitoring→Measurables
```

covered earlier in section 8.4.2.

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 8.5.4).

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

```
[rima92->monitoring->measurable]% list
type          name (key)           parameter  class                          producer
------------- -------------------- ---------- ------------------------------ -----------------
Enum          DeviceStatus                    Internal                       DeviceState
HealthCheck   ManagedServicesOk               Internal                       CMDaemonState
HealthCheck   Mon::Storage                    Internal/Monitoring/Storage    MonitoringSystem
Metric        nfs_v3_server_total             Disk                           NFS
Metric        nfs_v3_server_write             Disk                           NFS
...
```

The above example illustrates a list with some of the measurables that can be set for sampling on a newly installed system. A full list typically contains over two hundred 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
-------------------- ----------------------------
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
[myheadnode->monitoring->measurable]% show devicestatus
Parameter            Value
-------------------- ----------------------------
Class                Internal
Consolidator         none
Description          The device status
Disabled             no (DeviceState)
Gap                  0 (DeviceState)
Maximal age          0s (DeviceState)
Maximal samples      4,096 (DeviceState)
Name                 DeviceStatus
Parameter
Producer             DeviceState
Revision
Type                 Enum
[myheadnode->monitoring->measurable]% show diskspace
Parameter            Value
-------------------- ----------------------------
Class                Disk
Consolidator         - (diskspace)
Description          checks free disk space
Disabled             no (diskspace)
Gap                  0 (diskspace)
Maximal age          0s (diskspace)
Maximal samples      4,096 (diskspace)
```

```
Name                 diskspace
Parameter
Producer             diskspace
Revision
Type                 HealthCheck
```

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 8.5.1, 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 suggestions suggests over 200 possible objects:

Example

```
[rima92->monitoring->measurable]% show
Display all 221 possibilities? (y or n)
alertlevel:count     iotime:vda          mon::storage::engine::elements     oomkiller
alertlevel:maximum   iotime:vdb          mon::storage::engine::size         opalinkhealth
alertlevel:sum       ipforwdatagrams     mon::storage::engine::usage        packetsrecv:eth0
blockedprocesses     ipfragcreates       mon::storage::message::elements    packetsrecv:eth1
buffermemory         ipfragfails         mon::storage::message::size        packetssent:eth0
bytesrecv:eth0       ipfragoks           mon::storage::message::usage       packetssent:eth1
...
```

The single colon (":") indicates an extra parameter for that measurable.

Because there are a large number of metrics, it means that grepping a metrics list is sometimes handy.

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 shown in the first lines of the tab-completion suggestions of the show command of the previous example, shows up as alertlevel. However the list command displays it as AlertLevel. 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

```
[rima92->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.

Example

```
[rima92->monitoring->measurable]% has alertlevel:sum
rima92
node001
node002
[rima92->monitoring->measurable]% use devicesup
[rima92->monitoring->measurable[DevicesUp]]% has
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 8.5.1).

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 363) is used to set up a health check. The healthcheck can then be viewed from the `measurable` submode.

In the basic example of section 8.1, a trigger was set up from RIMA 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 RIMA 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 8.5.4.

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 | grep -i cpucheck
HealthCheckScript      cpucheck           1 / 222      <0 in submode>
```

The measurable submode shows that a measurable cpucheck is also created:

Example

```
[myheadnode->monitoring]% measurable list | grep -i cpucheck
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 8.5.4 to produce a fully configured health check.

8.5.4 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 RIMA View clickpath:

Monitoring→Data Producers

covered earlier in section 8.4.1.

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

```
[rima92->monitoring->setup]% list
Type             Name (key)       Arguments    Measurables  Node execution filters
---------------  ---------------  -----------  ------------ ----------------------
AggregateNode    AggregateNode                 8 / 222      <1 in submode>
AlertLevel       AlertLevel                    3 / 222      <1 in submode>
CMDaemonState    CMDaemonState                 1 / 222      <0 in submode>
ClusterTotal     ClusterTotal                  18 / 222     <1 in submode>
Collection       BigDataTools                  0 / 222      <2 in submode>
```

```
Collection      Cassandra                   0 / 222      <1 in submode>
...
```

In the preceding example, the AlertLevel data producer has 3 / 222 as the value for measurables. This means that this AlertLevel data producer provides 3 measurables out of the 222 configured measurables. They may be enabled or disabled, depending on whether the data producer is enabled or disabled, but they are provided in any case.

To clarify this point: if the list command is run from setup mode to list producers, then the producers that have configured measurables are the ones that have 1 or more as the numerator value in the Measurables column. Conversely, the data producers with 0 in the numerator of the Measurables column have no configured measurables, whether enabled or disabled, and are effectively just placeholders until the software for the data producers is installed.

So, comparing the list of producers in setup mode with the measurables in measurable mode:

Example

In measurable mode, the three AlertLevel measurables (the 3 out of 222) produced by the AlertLevel producer can be seen with:

```
[rima92->monitoring->measurable]% list | head -2; list | grep AlertLevel
Type       Name (key)     Parameter   Class        Producer
--------   ------------   ---------   ---------    -------------
Metric     AlertLevel     count       Internal     AlertLevel
Metric     AlertLevel     maximum     Internal     AlertLevel
Metric     AlertLevel     sum         Internal     AlertLevel
```

On the other hand, in measurable mode, there are no measurables seen for BigDataTools (the 0 out of 222) produced by the BigDataTools producer, when running, for example: list | head -2; list | grep BigDataTools.

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

```
[rima92->monitoring->setup]% list | head -2; list | grep mount
Type                   Name (key)     Arguments    Measurables  Node execution filters
---------------------- -------------- ------------ ------------ ----------------------
HealthCheckScript      mounts                      1 / 229      <0 in submode>
[rima92->monitoring->setup]% nodes mounts
node001..node003,rima92
```

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

```
[rima92->monitoring->setup]% show alertlevel
Parameter                      Value
------------------------------ ------------------------------------------------
Automatic reinitialize         yes
```

```
Consolidator                    default
Description                     Alert level as function of all trigger severities
Disabled                        no
Execution multiplexer           <1 in submode>
Fuzzy offset                    0
Gap                             0
Interval                        2m
Maximal age                     0s
Maximal samples                 4096
Measurables                     3 / 222
Name                            AlertLevel
Node execution filters          <1 in submode>
Notes                           <0 bytes>
Offset                          1m
Only when idle                  no
Revision
Type                            AlertLevel
When                            Timed
```

These properties are described in section 8.4.1. Most of these properties are inherited by the meaurables 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 3 further submodes (modes deeper than submodes are normally also just called submodes for convenience, rather than sub-submodes):

- `nodeexecutionfilters`
- `executionmultiplexers`
- `jobmetricsettings`

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 are of type `node`, `category`, `overlay`, `resource`, and `lua`. 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
AggregateNode   myhost
AggregatePDU    myhost
AlertLevel      myhost
Cassandra       Not used
```

```
ceph_global     Not used
chrootprocess   myhost
ClusterTotal    myhost
CMDaemonState   node001,node002,myhost
...
```

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 618) 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

```
[rima92->monitoring->setup[rogueprocess]]% nodeexecutionfilters
[rima92->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
------------------------------ ------------------------------------------------
Filter operation               Include
Name                           justhese
Nodes
Revision
Type                           Node
[...tup*[rogueprocess*]->nodeexecutionfilters*[justthese*]]% set nodes node001,node002
[...tup*[rogueprocess*]->nodeexecutionfilters*[justthese*]]% show
Parameter                      Value
------------------------------ ------------------------------------------------
Filter operation               Include
Name                           justhese
Nodes                          node001,node002
Revision
Type                           Node
[...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 node. In `cmsh`, a session to achieve this could be:

```
[rima92->monitoring->setup]% add procpidstat slapd
[rima92->monitoring->setup*[slapd*]]% set process slapd
[rima92->monitoring->setup*[slapd*]]% set consolidator none
[rima92->monitoring->setup*[slapd*]]% nodeexecutionfilters
[rima92->monitoring->setup*[slapd*]->nodeexecutionfilters]% add type headnodes
[rima92->monitoring->setup*[slapd*]->nodeexecutionfilters*[headnodes*]]% set headnode yes
[rima92->monitoring->setup*[slapd*]->nodeexecutionfilters*[headnodes*]]% show
[rima92->monitoring->setup*[slapd*]]% commit
```

The newly-defined slapd metric can now have its output displayed or plotted just like any other metric:

```
[rima92->device[rima92]]% latestmetricdata | grep slapd
MemoryUsed          slapd   Process   739 MiB      28.6s
SystemTime          slapd   Process   2m 35s       28.6s
ThreadsUsed         slapd   Process   50           28.6s
UserTime            slapd   Process   1h 36m       28.6s
VirtualMemoryUsed   slapd   Process   4.77 GiB     28.6s
```

- **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

```
[rima92->device[rima92]]% monitoringresources
Active
Docker::Host
Ethernet
Kubernetes::ApiServer
Kubernetes::ApiServerProxy
Kubernetes::Controller
Kubernetes::Node
Kubernetes::Proxy
Kubernetes::Scheduler
RDO
boot
...
```

  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 362 can be set to run on the active head node, by creating an arbitrary resource called myactive:

  Example

```
[rima92->monitoring->setup[cpucheck]]% nodeexecutionfilters
[...tup[cpucheck]->nodeexecutionfilters]% add resource "myactive"
[...tup*[cpucheck*]->nodeexecutionfilters*[myactive*]]% show
Parameter                      Value
------------------------------ ------------------------------------------------
Filter                         Include
Name                           myactive
Operator                       OR
Resources
Revision
Type                           Resource
```

  and then setting the Resources parameter to Active:

```
[...tup*[cpucheck*]->nodeexecutionfilters*[myactive*]]% set resources Active
[...tup*[cpucheck*]->nodeexecutionfilters*[myactive*]]% show
Parameter                      Value
------------------------------ ----------------------------------------------
Filter                         Include
Name                           myactive
Operator                       OR
Resources                      Active
Revision
Type                           Resource
[...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 8.6.7).

Execution multiplexer: A way to multiplex execution (have execution work elsewhere) for a data producer. It tells the cluster manager about the entities that the data producer is sampling for. A data producer runs and gathers data at the entity (node, category, lua, overlay, resource type) defined by the node execution filter, and with multiplex execution the data producer gathers samples from other entities. These entities can be nodes, categories, lua scripts, overlays, resources, and types. The entities from which it can sample are defined into groups called execution multiplexers. Execution multiplexers can thus be `node` multiplexers, `category` multiplexers, `lua` multiplexers, `type` multiplexers, `overlay` multiplexers, or `resource` multiplexers.

The `executionmultiplexers` mode can be entered for a data producer `dmesg` with:

Example

```
root@rima92 ~]# cmsh
[rima92]% 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 F.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

```
[rima92->monitoring->setup[dmesg]->executionmultiplexers]% add<TAB><TAB>
category  lua       node      overlay   resource  type
[rima92->monitoring->setup[dmesg]->executionmultiplexers]% add node nodes
[rima92->...*[dmesg*]->executionmultiplexers*[nodes*]]% show
Parameter                      Value
------------------------------ ----------------------------------------------
Filter operation               Include
Name                           nodes
Nodes
Revision
Type                           Node

[rima92->...*[dmesg*]->executionmultiplexers*[nodes*]]% set nodes node001,node002
[rima92->...*[dmesg*]->executionmultiplexers*[nodes*]]% show
Parameter                      Value
```

```
------------------------------ ----------------------------------------------
Filter operation               Include
Name                           nodes
Nodes                          node001,node002
Revision
Type                           Node
```

The concepts and expected behavior of node execution filters and execution multiplexers is covered in more explicit detail in Appendix J.

Job Metrics Settings: Job metrics settings are a submode for setting job metric collection options for the JobSampler data producer.

8.5.5 The standalone Submode

The standalone submode under the monitoring mode of cmsh allows entities that are not managed by the cluster manager to be configured for monitoring. This mode in cmsh corresponds to the RIMA View clickpath:

```
Monitoring→Standalone Monitored Entities
```

covered earlier in section 8.4.8.

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 cluster manager head node.

8.5.6 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 RIMA View clickpath:

```
Monitoring→Triggers
```

covered earlier in section 8.4.5.

By default, there are 3 triggers:

Example

```
[rima92->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 8.1, a trigger to run the killallyes script was configured using RIMA View.

The expression that was set for the killallyes script in the basic example using RIMA View can also be set in cmsh. For example:

Example

```
[rima92->monitoring->trigger]% add killallyestrigger
[rima92->monitoring->trigger*[killallyestrigger*]]% show
Parameter                      Value
------------------------------ ------------------------------------------------
Disabled                       no
During actions
Enter actions
Leave actions
Mark entity as failed          yes
Mark entity as unknown         no
Name                           killallyestrigger
Revision
Severity                       10
State flapping actions
State flapping count           5
State flapping period          5m
expression                     (*, *, *) == FAIL
[rima92->monitoring->trigger*[killallyestrigger*]]% expression
[rima92->monitoring->trigger*[killallyestrigger*]->expression[]]% show
Parameter                      Value
------------------------------ ------------------------------------------------
Entities
Measurables
Name
Operator                       EQ
Parameters
Revision
Type                           MonitoringCompareExpression
Use raw                        no
Value                          FAIL
[rima92->monitoring->trigger*[killallyestrigger*]->expression[]]% set entities rima92
[rima92->monitoring->trigger*[killallyestrigger*]->expression*[*]]% set measurables CPUUser
[rima92->monitoring->trigger*[killallyestrigger*]->expression*[*]]% set operator GT
[rima92->monitoring->trigger*[killallyestrigger*]->expression*[*]]% set value 50
[rima92->monitoring->trigger*[killallyestrigger*]->expression*[*]]% commit
[rima92->monitoring->trigger*[killallyestrigger*]->expression*[*]]% set name killallyesexp
Field                  Message
---------------------- ------------------------------------------------------
actions                Warning: No actions were set
============================== killallyestrigger ==============================
[rima92->monitoring->trigger[killallyestrigger]->expression[killallyesexp]]% exit
[rima92->monitoring->trigger[killallyestrigger]->expression]% exit
[rima92->monitoring->trigger[killallyestrigger]]% set enteractions killallyesname
[rima92->monitoring->trigger*[killallyestrigger*]]% commit
[rima92->monitoring->trigger[killallyestrigger]]%
```

The expression format is shown in `cmsh` as:

(*<entity>*, *<measurable>*, *<parameter>*) *<comparison operator>* *<value>*

Here:

- an entity, as described in section 8.2.1, 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 "rima92|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 8.2.1) 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 like 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 8.5.3 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 8.1. In this section the task is continued and completed as follows:

```
[rima92->monitoring->trigger]% expression killallyestrigger
[...trigger[killallyestrigger]->expression[killallyesexp]]% get measurables
CPUUser
[...trigger[killallyestrigger]->expression[killallyesexp]]% set measurables cpucheck
[...trigger*[killallyestrigger*]->expression*[killallyesexp*]]% commit
```

8.6 Obtaining Monitoring Data Values

The monitoring data values that are logged by devices can be used to generate graphs using the methods in section 8.3. 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.

8.6.1 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 8.5.3). 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

```
[rima92->device]% enummetrics node001
Type         Name               Parameter  Class                        Producer
------------ ------------------ ---------- ---------------------------- ----------------
Enum         DeviceStatus                  Internal                     DeviceState
[rima92->device]% use rima92
[rima92->device[rima92]]% measurables
Type         Name               Parameter  Class                        Producer
------------ ------------------ ---------- ---------------------------- ----------------
Enum         DeviceStatus                  Internal                     DeviceState
HealthCheck  ManagedServicesOk             Internal                     CMDaemonState
HealthCheck  Mon::Storage                  Internal/Monitoring/Storage  MonitoringSystem
...
[rima92->device[rima92]]% exit
[rima92->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 8.5.3.

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

```
[rima92->device]% partition use base
[rima92->partition[base]]% measurables
Type         Name                Parameter  Class                       Producer
------------ ------------------- ---------- --------------------------- ----------------
Metric       CoresTotal                     Total                       ClusterTotal
Metric       CoresUp                        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.

8.6.2 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

```
[rima92->device]% use rima92
[rima92->device[rima92]]% samplenow
Measurable          Parameter Type      Value        Age     Info
------------------- --------- --------- ------------ ------- -----
AlertLevel          count     Internal  0            2.01s
AlertLevel          maximum   Internal  0            2.01s
AlertLevel          sum       Internal  0            2.01s
BlockedProcesses              OS        0 processes  2.01s
BufferMemory                  Memory    847 KiB      2.01s
...
```

The entity used can also be in other modes that have measurables, such as the base partition (output truncated):

Example

```
[rima92->device]% partition use base
[rima92->partition[base]]% samplenow
Measurable       Parameter    Type         Value      Age        Info
---------------- ------------ ------------ ---------- ---------- ----------
CoresTotal                    Total        24         0.001s
CoresUp                       Total        24         0.001s
DevicesClosed                 Total        0          0.001s
DevicesDown                   Total        0          0.001s
DevicesTotal                  Total        0          0.001s
DevicesUp                     Total        0          0.001s
...
```

The -n|--nodes Option

The -n option is used to sample specified nodes or node ranges:

Example

```
[rima92->partition[base]]% device
[rima92->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

```
[rima92->device]% samplenow --metrics loadone loadfifteen --n node001,node002
Entity       Measurable         Parameter Type      Value   Age     Info
------------ ------------------ --------- --------- ------- ------- -----
node001      LoadOne                      OS        0.04    0.08s
node002      LoadOne                      OS        0       0.077s
[rima92->device]% samplenow --checks -n node001..node002
Entity       Measurable         Parameter Type      Value   Age     Info
------------ ------------------ --------- --------- ------- ------- -----
node001      ManagedServicesOk            Internal  PASS    0.177s
node001      defaultgateway               Network   PASS    0.145s
node001      diskspace                    Disk      PASS    0.16s
node001      dmesg                        OS        PASS    0.177s
[rima92->device]% samplenow --checks diskspace -n node001..node002
Entity       Measurable         Parameter Type      Value   Age     Info
------------ ------------------ --------- --------- ------- ------- -----
node001      diskspace                    Disk      PASS    0.095s
node002      diskspace                    Disk      PASS    0.097s
[rima92->device]%
```

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

```
[rima92->device[node001]]% samplenow ntp
Measurable   Type         Value      Age        Info
------------ ------------ ---------- ---------- ----------
ntp          Internal     PASS       0.51s
[rima92->device[node001]]% samplenow --debug ntp
Measurable   Type         Value      Age        Info
------------ ------------ ---------- ---------- ------------------
ntp          Internal     PASS       0.524s     command: "ps -e"+
[rima92->device[node001]]% samplenow --debug -v ntp
Measurable   Type         Value      Age        Info
------------ ------------ ---------- ---------- ----------------------------------------
ntp          Internal     PASS       0.543s     command: "ps -e"
                                                ntpd process found, pid: 11226
                                                command: "/sbin/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 : 1586171027.783
                                                time on node             : 1586171027.771
                                                time difference          : 0.012
                                                execution time 0.06

[rima92->device[node001]]% !service ntpd stop
Redirecting to /bin/systemctl stop ntpd.service

[rima92->device[node001]] samplenow --debug -v ntp
measurable   Type         Value      Age        Info
------------ ------------ ---------- ---------- --------------------
ntp          Internal     UNKNOWN    10s        timed out after: 10s
```

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

```
[rima92->device]% samplenow -s UP
Entity       Measurable         Parameter Type      Value       Age    Info
------------ ------------------ --------- --------- ----------- ------ -----
rima92     AlertLevel         count     Internal  0           4.67s
rima92     AlertLevel         maximum   Internal  0           4.67s
rima92     AlertLevel         sum       Internal  0           4.67s
rima92     BlockedProcesses             OS        0 processes 4.67s
rima92     BufferMemory                 Memory    847 KiB     4.67s
rima92     BytesRecv          eth0      Network   357 MiB     4.67s
rima92     BytesRecv          eth1      Network   78.7 MiB    4.67s
...
```

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

```
[rima92->device]% samplenow -s UP -n node001
Measurable         Parameter Type      Value       Age    Info
------------------ --------- --------- ----------- ------ -----
AlertLevel         count     Internal  0           0.081s
AlertLevel         maximum   Internal  0           0.081s
AlertLevel         sum       Internal  0           0.081s
...
```

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 8.6.3) 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 8.6.4) displays monitoring data gathered over a period of time in a variety of formats.

8.6.3 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 374) 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 374) 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 373) 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

```
[rima92->device]% use node001
[rima92->device[node001]]% latestmetricdata
Measurable         Parameter  Type          Value          Age         Info
------------------ ---------- ------------ ---------------------- ---------------
AlertLevel         count      Internal      0              1m 12s     FAIL schedulers
AlertLevel         maximum    Internal      0              1m 12s     FAIL schedulers
AlertLevel         sum        Internal      0              1m 12s
```

```
BlockedProcesses             OS          0 processes   1m 12s
BufferMemory                 Memory      847 KiB       1m 12s
BytesRecv         eth0       Network     311.611 B/s   1m 12s
BytesRecv         eth1       Network     0 B/s         1m 12s
BytesSent         eth0       Network     349.953 B/s   1m 12s
BytesSent         eth1       Network     0 B/s         1m 12s
CPUGuest                     CPU         0 Jiffies/s   1m 12s
...
```

Valid entity grouping options and other options can be seen in the help text for the `latestmetricdata` and `latesthealthdata` commands.

Example

```
[rima92->device]% help latestmetricdata
Name: Latestmetricdata - Display the latest metric data

Usage: latestmetricdata [OPTIONS] [<entity>]

Options:
    -v, --verbose
        Be more verbose

    -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
...
```

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

```
[rima92->device]% latestmetricdata -g mynodegroup
Entity     Measurable          Parameter  Type        Value          Age     State  Info
---------- ------------------- ---------- ----------- -------------- ------- ------ ------
node001    AlertLevel          count      Internal    0              15.4s
node001    AlertLevel          maximum    Internal    0              15.4s
node001    AlertLevel          sum        Internal    0              15.4s
node001    BlockedProcesses               OS          0 processes    1m 41s
node001    BufferMemory                   Memory      27.3 KiB       41s
node001    BytesRecv           ens3       Network     622.722 B/s    56s
node001    BytesSent           ens3       Network     580.088 B/s    56s
...
node002    AlertLevel          count      Internal    0              15.4s
node002    AlertLevel          maximum    Internal    0              15.4s
node002    AlertLevel          sum        Internal    0              15.4s
node002    BlockedProcesses               OS          0 processes    1m 20s
node002    BufferMemory                   Memory      39 KiB         2m 20s
node002    BytesRecv           ens3       Network     696.364 B/s    35.6s
node002    BytesSent           ens3       Network     574.08 B/s     35.6s
...
```

and the entities that are displayed, for example, in nodegroup mode:

Example

```
[rima92->nodegroup]% latestmetricdata mynodegroup
Measurable   Parameter    Type         Value      Age         State   Info
------------ ------------ ------------ ---------- ---------- ------- -----
CoresTotal                Total        4          31.8s
CoresUp                   Total        4          31.8s
FPGAsTotal                Total        0          31.8s
FPGAsUp                   Total        0          31.8s
GPUsTotal                 Total        0          31.8s
GPUsUp                    Total        0          31.8s
NodesClosed               Total        0          31.8s
NodesDown                 Total        0          31.8s
NodesTotal                Total        2          31.8s
NodesUp                   Total        2          31.8s
```

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

```
[rima92->device]% latestmonitoringcounters node001
Measurable       Parameter    Type         Value                Age         Info
---------------- ------------ ------------ -------------------- ---------- -----
BytesRecv        eth0         Network      286 MiB              11.7s
BytesRecv        eth1         Network      0 B                  11.7s
BytesSent        eth0         Network      217 MiB              11.7s
BytesSent        eth1         Network      0 B                  11.7s
CPUGuest                      CPU          0 Jiffies/s          11.7s
CPUIdle                       CPU          60.1 Jiffies/s       11.7s
CPUIrq                        CPU          0 Jiffies/s          11.7s
CPUNice                       CPU          66 Jiffies/s         11.7s
...
```

The reader can compare the preceding example output against the example output of the latestmetricdata command (page 376) to become familiar with the meaning of cumulative output.

8.6.4 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.

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 8.6.1).
- 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 to the fixed end time.
    - For the <*end-time*>, a number prefixed with "+" is used. It indicates a time that much later to 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

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
-i, --intervals	<*number*>	number of samples to show
-u, --unix-epoch		use a unix timestamp instead of using the default date format
-d, --delimiter	"<*string*>"	set the delimiter to a character
--sum		sum over specified entities
--max		maximum over specified entities
--min		minimum over specified entities
--avg		average over specified entities
--raw		show the metric value without units
--human		show the metric value with units (default)
--consolidationinterval		retrieve data from the consolidator with specified interval
--consolidationoffset		retrieve data from the consolidator with specified (interval, offset)
--timeaverage		calculate the average for the entire interval for specified devices
--delta		display change relative to previous value
--clip		clip data samples to the requested interval
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

```
[rima92->device[node001]]% dumpmonitoringdata 18:00:00 18:02:00 loadone
Timestamp                  Value      Info
-------------------------- ---------- ----------
2017/08/30 17:58:00        0.02
2017/08/30 18:00:00        0.01
2017/08/30 18:02:00        0.02
```

Double quotes are needed for times with a YY/MM/DD specification:

Example

```
[rima92->device[node001]]% dumpmonitoringdata "17/08/30 18:00" "17/08/30 18:02" loadone
Timestamp                  Value      Info
-------------------------- ---------- ----------
2017/08/30 17:58:00        0.02
2017/08/30 18:00:00        0.01
2017/08/30 18:02:00        0.02
```

Unix epoch time can also be set:

Example

```
[rima92->device[node001]]% !date -d "Aug 30 18:00:00 2017" +%s
1504108800
[rima92->device[node001]]% dumpmonitoringdata 1504108800 1504108920 loadone
Timestamp                  Value      Info
-------------------------- ---------- ----------
2017/08/30 17:58:00        0.02
2017/08/30 18:00:00        0.01
2017/08/30 18:02:00        0.02
```

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

```
[rima92->device]% dumpmonitoringdata -i 0 -10m now loadone node001
Timestamp                  Value      Info
-------------------------- ---------- ----------
2017/07/21 14:56:00        0.01
2017/07/21 14:58:00        0.14
2017/07/21 15:00:00        0.04
2017/07/21 15:02:00        0.04
2017/07/21 15:04:00        0.08
2017/07/21 15:06:00        0.08
```

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

```
[rima92->device]% dumpmonitoringdata -i 3 -10m now loadone node001
Timestamp                  Value      Info
-------------------------- ---------- ----------
2017/07/21 21:49:36        0
2017/07/21 21:54:36        0.0419998
2017/07/21 21:59:36        no data
```

A set of nodes can be specified for the dump:

```
[rima92->device]% dumpmonitoringdata -n node001..node002 -5m now cpuidle
Entity       Timestamp                  Value             Info
------------ -------------------------- ----------------- ----------
node001      2017/07/20 20:14:00        99.8258 Jiffies/s
```

```
node001      2017/07/20 20:16:00        99.8233 Jiffies/s
node001      2017/07/20 20:18:00        99.8192 Jiffies/s
node001      2017/07/20 20:20:00        99.8475 Jiffies/s
node002      2017/07/20 20:14:00        99.7917 Jiffies/s
node002      2017/07/20 20:16:00        99.8083 Jiffies/s
node002      2017/07/20 20:18:00        99.7992 Jiffies/s
node002      2017/07/20 20:20:00        99.815 Jiffies/s
[rima92->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

```
[rima92->device]% dumpmonitoringdata -2h now -i 3 loadone -n node00[1-2] --sum
Timestamp                  Value      Info
-------------------------- ---------- ----------
2017/07/20 18:30:27        0.0292462
2017/07/20 19:30:27        0
2017/07/20 20:30:27        no data
```

Each entry in the values 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

```
[rima92->device]% dumpmonitoringdata -2h now -i 3 loadone -n node00[1-2]
Entity       Timestamp                  Value      Info
------------ -------------------------- ---------- ----------
node001      2017/07/20 18:30:27        0
node001      2017/07/20 19:30:27        0
node001      2017/07/20 20:30:27        no data
node002      2017/07/20 18:30:27        0.0292462
node002      2017/07/20 19:30:27        0
node002      2017/07/20 20:30:27        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

```
[rima92->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
Entity       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

```
[rima92->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

```
[rima92->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
Entity       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

```
[rima92->device[node001]]% dumpmonitoringdata -5m now bytessent:eth0 --clip
# Start : 1552400295 / Tue Mar 12 14:18:15 2019
# End   : 1552400595 / Tue Mar 12 14:23:15 2019
Timestamp                  Value        Info
-------------------------- ------------ ----------
2019/03/12 14:18:15        201.942 B/s
2019/03/12 14:20:15        217.883 B/s
2019/03/12 14:22:15        233.058 B/s
2019/03/12 14:23:15        235.831 B/s
```

In the preceding example, the first 3 samples are raw samples, the last sample is an interpolated value, over a time period evaluated as being from `14:18:15` to `14:23:15`. The epoch times for this period, and corresponding human-readable values are shown in 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 8.2.2).

Example

```
[rima92->device]% dumpmonitoringdata  -2m now loadone -s up
Entity       Timestamp                  Value      Info
------------ -------------------------- ---------- ----------
rima92     2017/07/21 15:00:00        0.35
rima92     2017/07/21 15:02:00        0.53
node001      2017/07/21 14:12:00        0.04
node001      2017/07/21 15:02:00        0.04
node002      2017/07/21 15:00:00        0.22
node002      2017/07/21 15:02:00        0.21
[rima92->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_callinginit`, `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

```
[rima92->device[node001]]% dumpmonitoringdata --delta -6m now pageout
Timestamp                  Value        Delta            Info
-------------------------- ------------ ---------------- ----------
2018/09/10 17:49:28        1015.46 B/s  nan
2018/09/10 17:51:28        1.35 KiB/s   2.7 B/s/s
2018/09/10 17:53:28        1.34 KiB/s   -0.083 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

```
[rima92->device[rima92]]% dumpmonitoringdata -2h now memoryused:cmd -n node001 --delta
Timestamp                  Value      Delta          Info
-------------------------- ---------- -------------- ----------
2018/08/15 10:00:00.812    68.4 MiB   nan
2018/08/15 10:02:00.812    68.4 MiB   0.0341333 B/s
2018/08/15 12:10:00.812    68.4 MiB   0 B/s
```

The roughly 0B/s increase over 2 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

```
[rima92->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
[rima92->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

```
[rima92->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
[rima92->device]%
```

For an intersection of sets, the only syntax allowed is one that uses different grouping options:

Example

```
[rima92->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.

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

```
[rima92->device[node001]]% dumpmonitoringdata -10m now threadsused:cmd
Timestamp                  Value      Info
-------------------------- ---------- ----------
2017/07/21 11:16:00        42
2017/07/21 11:20:00        42
2017/07/21 11:22:00        41
2017/07/21 11:24:00        42
2017/07/21 11:26:00        42
```

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 11:18:00 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

```
[rima92->softwareimage]% dumpmonitoringdata  -10m now nodesup default-image
Timestamp                Value      Info
------------------------ ---------- ----------
2017/07/13 16:43:00      2
2017/07/20 17:37:00      2
[rima92->softwareimage]%
```

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 about 7 days because the nodes happened to be booted then.

8.6.5 Monitoring Data Health Overview–The `healthoverview` Command

In figure 8.20, section 8.4.6, the RIMA View clickpath

```
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

```
[rima92->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
```

8.6.6 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

```
[rima92->device]% monitoringinfo  -n node001
Service                     Queued Handled  Cache miss Stopped Suspended Last operation
--------------------------- ------ -------- ---------- ------- --------- -------------------
Mon::CacheGather            0      0        0          yes     no        -
Mon::DataProcessor          0      0        0          yes     no        -
Mon::DataTranslator         0      932,257  0          no      no        Mon Jul 24 11:34:00
Mon::EntityMeasurableCache  0      0        0          no      no        Thu Jul 13 16:39:52
Mon::MeasurableBroker       0      0        0          no      no        -
Mon::Replicate::Collector   0      0        0          yes     yes       -
Mon::Replicate::Combiner    0      0        0          yes     yes       -
Mon::RepositoryAllocator    0      0        0          yes     no        -
Mon::RepositoryTrim         0      0        0          yes     no        -
Mon::TaskInitializer        0      30       0          no      no        Thu Jul 13 16:39:52
Mon::TaskSampler            30     233,039  0          no      no        Mon Jul 24 11:34:00
Mon::Trigger::Actuator      0      0        0          yes     no        -
Mon::Trigger::Dispatcher    0      0        0          yes     no        -

Cache                   Size         Updates      Requests
----------------------- ------------ ------------ ------------
ConsolidatorCache       0            17           0
EntityCache             10           17           935,280
GlobalLastRawDataCache  87           17           0
LastRawDataCache        142          17           427,301
MeasurableCache         231          17           935,230

Cache             Up           Down         Closed
----------------- ------------ ------------ ------------
DeviceStateCache  3            0            0

Replicator               First        Last         Requests     Samples      Sources
------------------------ ------------ ------------ ------------ ------------ ------------
ReplicateRequestHandler  -            -            0            0

Cache        Queued       Delivered    Handled      Pickup
------------ ------------ ------------ ------------ ------------
Cache        0            120          932,257      7,766

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%     -
```

8.6.7 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 RIMA 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@rima92 ~]# cmsh
[rima92]% monitoring setup use ssh2node
[...->monitoring->setup[ssh2node]]% executionmultiplexers
[...->executionmultiplexers]% show
Type                             None
[...->executionmultiplexers]% use all nodes
[...->executionmultiplexers[All nodes]]% get types
Node
[...->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@rima92 ~]# cmsh
[rima92]% 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@rima92 ~]# cmsh
[rima92]% 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.

8.6.8 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 533).

Disabling job information collection completely: by setting the value of the `JobInformationDisabled` directive to `0` (page 537).

For the Slurm workload manager only, disabling job accounting: by setting the value of the `SlurmDisableAccountingParsing` directive to `0` (page 532).

Reducing the duration for which job data is stored: by reducing the value of the `JobInformationKeepDuration` (page 537).

8.6.9 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

```
[rima92]% device
[rima92->device]% monitoringpickup
Hostname      Interval   Times    Priority
------------- ---------- -------- ----------
rima92      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

```
[rima92]% device use node001
[rima92->device[node001]]% monitoringpickup --interval 1m
Changed 1 pickup intervals
[rima92->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

```
[rima92]% device use node001
[rima92->device[node001]]% monitoringpickup --interval 1m --times 10
Changed 1 pickup intervals
[rima92->device[node001]]% monitoringpickup
Hostname      Interval    Times    Priority
------------- ----------- -------- ---------
node001       1m          10       100
```

The `--forever` option lets the pickup be carried out "forever"[3].

Example

```
[rima92->device[node001]]% monitoringpickup --interval 30s --forever
Changed 1 pickup intervals
[rima92->device[node001]]% monitoringpickup
Hostname      Interval    Times    Priority
------------- ----------- -------- ---------
node001       30s         -        100
```

The `--priority` option applies the priority to equal or lower priority settings:

Example

```
[rima92]% device
[rima92->device]% monitoringpickup -n node00[1-2]
Hostname      Interval    Times    Priority
------------- ----------- -------- ---------
node001       1m          12       80
node002       1m          17       20
[rima92->device]% monitoringpickup -n node00[1-2] --interval 5s --priority 50
Changed 1 pickup intervals
[rima92->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:

```
[rima92->device]% monitoringpickup -n node00[1-2]
Hostname      Interval    Times    Priority
------------- ----------- -------- ---------
node001       1m          12       80
node002       2m          0        0
```

[3] Strictly speaking, "forever" means $(2^{64} - 1)$ times on the 64-bit architecture that the cluster manager runs on. For comparison, $(2^{64} - 1)$ seconds is about 585 billion years.

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`.

```
[rima92->...->jobmetricsettings]% show
Parameter            Value
-------------------- ------
...
Pickup interval      5s
Pickup priority      50
Pickup times         12
```

8.7 Offloaded Monitoring

Offloaded monitoring is a feature introduced in RIMA Cluster Manager version 9.1.

Traditional cluster manager 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 measurables are running.

Offloaded monitoring in the cluster manager 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 8.7.3.

8.7.1 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, the cluster manager simply implements it as a role that is assigned to nodes. The cluster manager backend then manages the details of offloaded monitoring.

8.7.2 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

```
[rima92->device]% use node001
[rima92->device[node001]]% roles
[rima92->device[node001]->roles]% assign monitoring
[rima92->device*[node001*]->roles*[monitoring*]]% show
Parameter                      Value
------------------------------ ------------------------------------------------
Name                           monitoring
Revision
Type                           MonitoringRole
Add services                   yes
Provisioning associations      <0 internally used>
Number of backups              2
Backup ring                    automatic
[rima92->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.

8.7.3 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.

The cluster manager 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 the cluster manager is configured with two head nodes as described in Chapter 10. 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 RIMA Cluster Manager 9.1, in comparison with a head node pair in RIMA Cluster 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 216), 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):

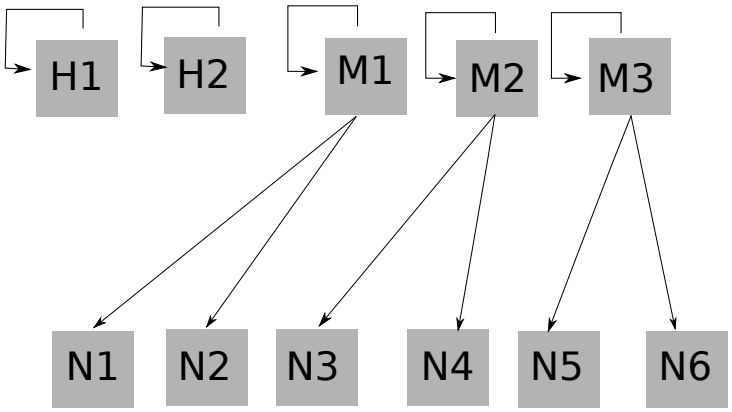


Figure 8.27: Monitoring Sampling Flow For Offloaded Monitoring With Dedicated Monitoring Nodes

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 8.7.2) is used to configure the number of backups. In the following schematic, two neighboring monitoring nodes are used as backup:

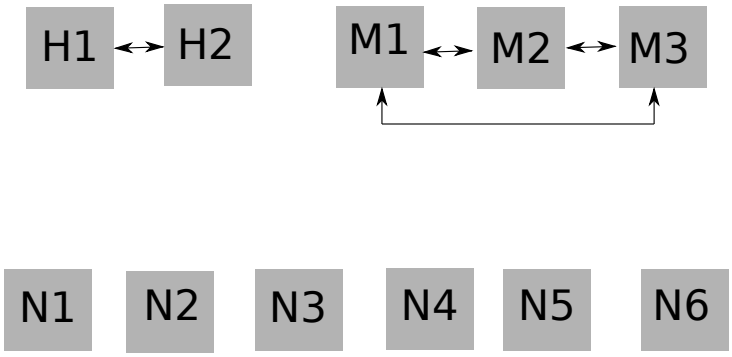


Figure 8.28: A Simple Backup Flow For Offloaded Monitoring With Dedicated Monitoring Nodes

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 RIMA 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:

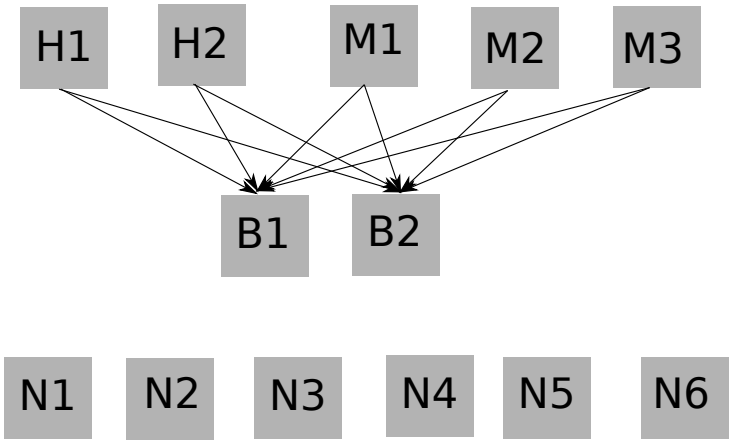


Figure 8.29: A More Sophisticated Backup Flow For Offloaded Monitoring With Dedicated Monitoring Nodes And Dedicated Backup Nodes

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-availablity configuration, have the monitoring sampling data flow shown by the following schematic (figure 8.30):

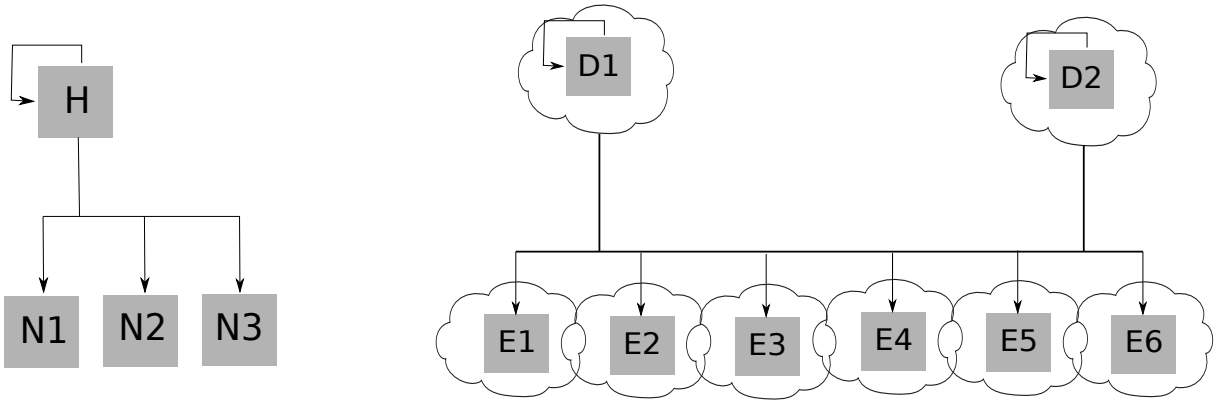


Figure 8.30: Sampling Flow For Offloaded Monitoring With Non-HA Edge Director Nodes

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 8.31):

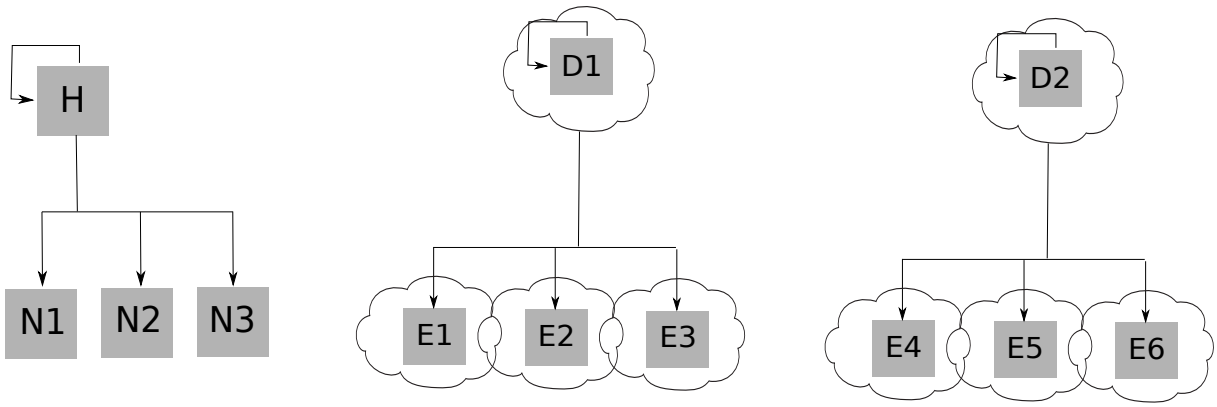


Figure 8.31: Sampling Flow For Offloaded Monitoring With HA Edge Director Nodes

The backup data flow for a non-HA configuration would then be as follows for an edge director (figure 8.32):

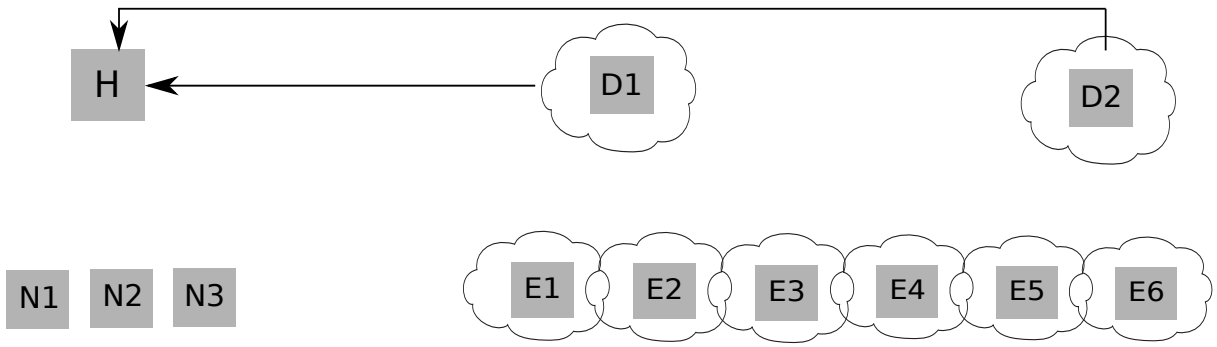


Figure 8.32: Backup Flow For Offloaded Monitoring With Non-HA Edge Director Nodes

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 8.33):

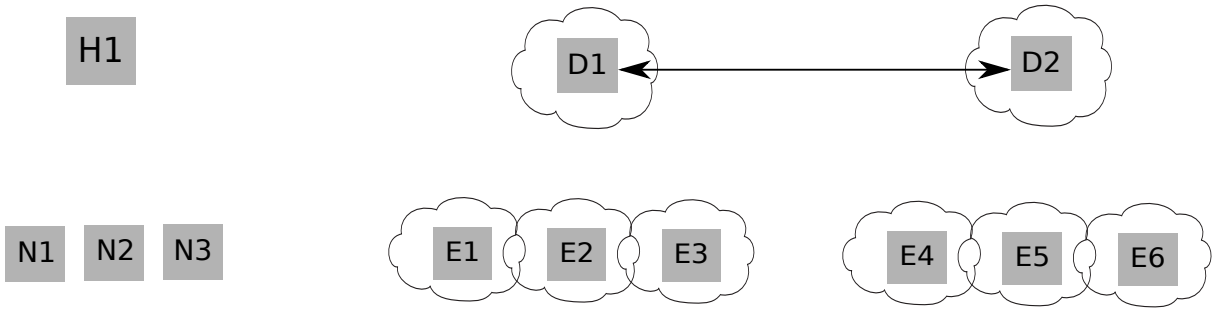


Figure 8.33: Backup Flow For Offloaded Monitoring With HA Edge Director Nodes

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

8.8 The User Portal

The user portal is a restricted version of RIMA 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 $\oplus$ 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@rima92 ~]# yum install cm-webportal
...
Is this ok [y/N]: y
Downloading Packages:
...
Complete!
```

8.8.1 Accessing The User Portal

The user portal is compatible with most browsers using reasonable settings, and is supported for the same browsers that RIMA 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 aforementioned link icon within the colored user portal block of the head node landing page.

- 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 warning about the site certificate being untrusted appears.

The certificate is a self-signed certificate (the X509v3 certificate of Chapter 4 of the *Installation Manual*), generated and signed by RIMA, and the attributes of the cluster owner are part of the certificate. However, RIMA is not a recognized Certificate Authority (CA) like the CAs that are recognized by a browser, which is why the warning appears.

The SSL configuration file is located at `/etc/httpd/conf.d/ssl.conf`. Within `ssl.conf`, by default the PEM-encoded server certificate is set to `/etc/pki/tls/certs/localhost.crt`. The entry should be changed if generating and installing the certificate elsewhere. A key-based certificate can be used instead, with a private key then expected by the `ssl.conf` file at `etc/pki/tls/private/localhost.key`.

For a portal that is not accessible from the outside world, such as the internet, the warning about RIMA not being a recognized Certificate Authority is not an issue, and the user can simply accept the "untrusted" certificate.

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.

8.8.2 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@rima92 ~]# 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 RIMA View

  Example

  ```
  [root@rima92 ~]# cmsh
  [rima92]% user
  [rima92->user]% add forrestgump
  [rima92->user[forrestgump]]% set password
  enter new password:
  retype new password:
  [rima92->user*[forrestgump*]]% commit
  [rima92->user[forrestgump]]% quit
  ```

- the common username and password should be set in the appropriate place in the configuration file, `common-credentials.json`:

Example

```
[root@rima92 ~]# 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 RIMA 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.

8.8.3 User Portal Access

By default, the user profile (section 6.4) is set to `readonly`, which allows viewing of the information presented in the user portal, without allowing it to be altered.

8.8.4 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 8.34):

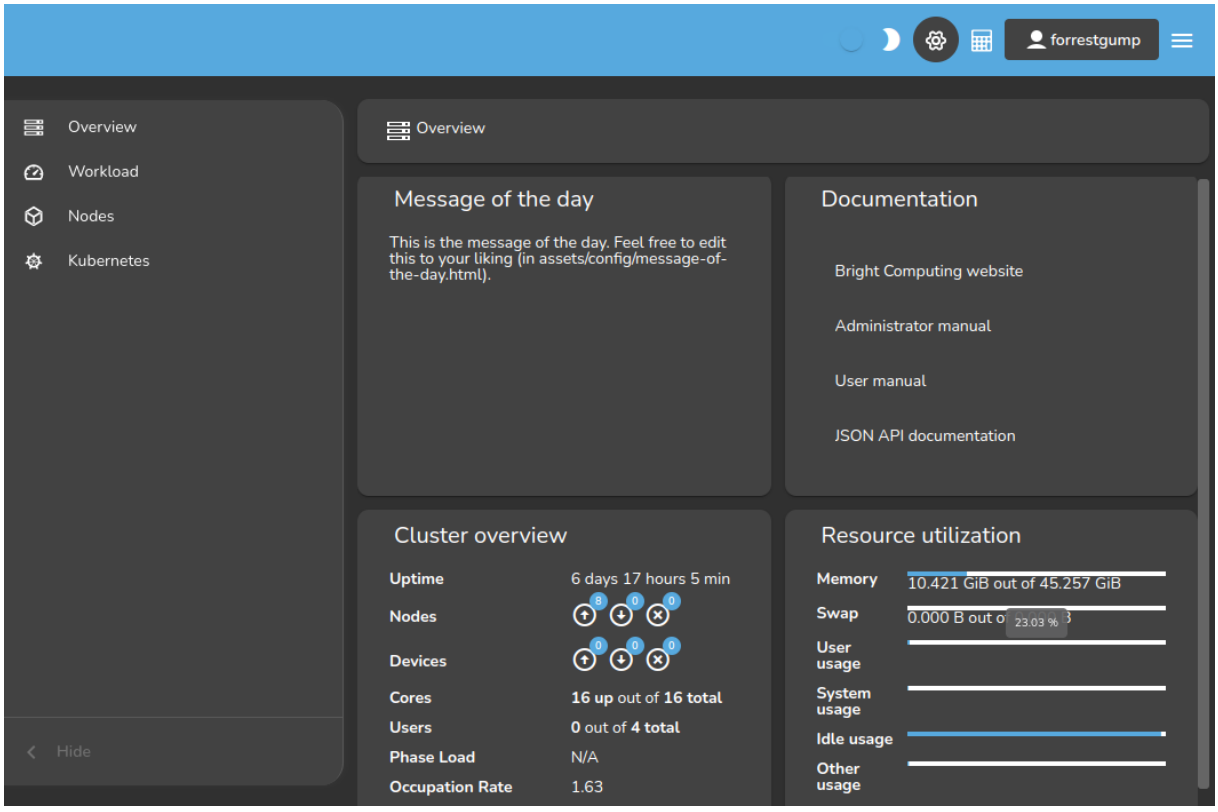


Figure 8.34: User Portal: Overview Page

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 RIMA View is then displayed.

The accounting and reporting page allows job accounting to be viewed in an accounting panel.

8.9 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 RIMA Cluster 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. 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, the cluster manager 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

8.10 Event Viewer

Monitoring in the cluster manager 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 the cluster manager 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.

the cluster manager events can be handled and viewed in several ways.

8.10.1 Viewing Events In RIMA View

In RIMA View, events can be viewed by clicking on the Events icon of figure 8.5. This opens up a window with a sortable set of columns listing the events in the events log, and with by default with the most recent events showing up first.

8.10.2 Viewing Events In cmsh

The events command is a global cmsh command. It allows events to be viewed at several severity levels (section 8.2.7), 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 level <level>
          events clear
          events details <id> [<id>]
          events <number> [level]
          events follow

Arguments:

 level info,notice,warning,error,alert
```

Running the command without any option shows event settings, and displays any event messages that have not been displayed yet in the session:

Example

```
[rima92->device]% events
Private events:   off
Broadcast events: on
Level:            notice
custom .....................[   RESET   ]  node001
```

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 RIMA 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 RIMA 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.

- 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 a specific event with the index value of <*id*>, which is a number that refers to an 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

```
[rima92->softwareimage]% events on private
Private events:   on
[rima92->softwareimage]% updateprovisioners
Provisioning nodes will be updated in the background.
[rima92->softwareimage]%
Tue Apr 29 01:19:12 2014 [notice] rima92: Provisioning started: sendi...
[rima92->softwareimage]%
Tue Apr 29 01:19:52 2014 [notice] rima92: Provisioning completed: sen...
updateprovisioners [ COMPLETED ]
[rima92->softwareimage]% !#events were indeed seen in cmsh session
[rima92->softwareimage]% !#now block the events and rerun update:
[rima92->softwareimage]% events off private
```

```
Private events:   off
[rima92->softwareimage]% updateprovisioners
Provisioning nodes will be updated in the background.
[rima92->softwareimage]% !#let this 2nd update run for a while
[rima92->softwareimage]% !#(time passes)
[rima92->softwareimage]% !#nothing seen in cmsh session.
[rima92->softwareimage]% !#show a 2nd update did happen:
[rima92->softwareimage]% events 4 | grep -i provisioning
Tue Apr 29 01:19:12 2014 [notice] rima92: Provisioning started: sendi...
Tue Apr 29 01:19:52 2014 [notice] rima92: Provisioning completed: sen...
Tue Apr 29 01:25:37 2014 [notice] rima92: Provisioning started: sendi...
Tue Apr 29 01:26:01 2014 [notice] rima92: Provisioning completed: sen...
```

8.10.3 Using The Event Bucket From The Shell For Events And For Tagging Device States

Event Bucket Default Behavior

The cluster manager *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`, and a message can be written to the event pane like this:

Example

```
[root@rima92 ~]# echo "Some text" > /var/spool/cmd/eventbucket
```

This adds an event with, by default, the `info` severity level, to the event pane, with the *InfoMessage* "Some text".

8.10.4 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

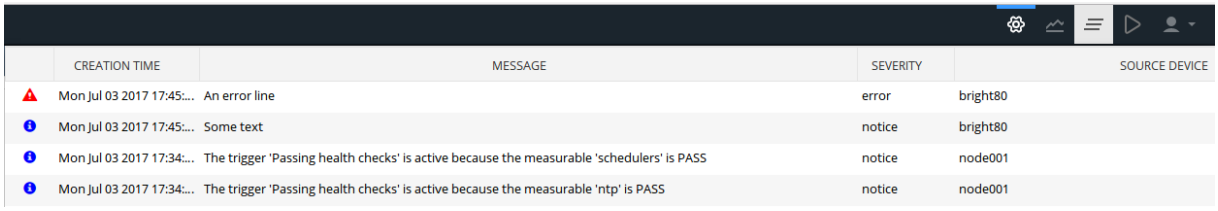
To write events at specific severity levels (section 8.2.7), and not just at the `info` level, the appropriate text can be prepended from the following to the text that is to be displayed:

```
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 RIMA View event viewer as shown in figure 8.35:



	CREATION TIME	MESSAGE	SEVERITY	SOURCE DEVICE
⚠	Mon Jul 03 2017 17:45:...	An error line	error	bright80
ℹ	Mon Jul 03 2017 17:45:...	Some text	notice	bright80
ℹ	Mon Jul 03 2017 17:34:...	The trigger 'Passing health checks' is active because the measurable 'schedulers' is PASS	notice	node001
ℹ	Mon Jul 03 2017 17:34:...	The trigger 'Passing health checks' is active because the measurable 'ntp' is PASS	notice	node001

Figure 8.35: RIMA 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, `rima92`, 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 rima92"
rima92 .................. (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
```

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 RIMA 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
   ```

8.11 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 engoineer 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. The cluster manager makes the results available in the `locations` submode of the `base` partition of `cmsh`:

Example

```
[rima92->partition[base]]% locations
Type             Entity                   Age      Latitude    Longitude   Height   Message
---------------- ------------------------ -------- ----------- ----------- -------- ----------------
EdgeSite         Amsterdam West           2d 17h   52.1904     4.939       0        Amsterdam-South
EdgeSite         Fort-Collins             6d 7h    40.5538     -105.0849   0        Fort-Collins
HeadNode         rima                     57m 3s   52.3927     4.8361      0        Amsterdam
PhysicalNode     bright-office-director   2d 16h   52.1903     4.9163      0        Amsterdam-South
PhysicalNode     bright-office-node001    2d 17h   52.1904     4.9617      0        Amsterdam-South
PhysicalNode     fort-collins-director    6d 7h    40.5538     -105.0849   0        Fort-Collins
[rima92->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.

The locations can be viewed in RIMA View via NOC mode (page 332).

8.12 Monitoring Report Queries

8.12.1 Monitoring Report Queries In cmsh

There are usually several hundred sources of data in the cluster manager, 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 the cluster manager 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

```
[rima92->monitoring->report]% fields | head -30
Name                             Type         Values
-------------------------------- ------------ ---------------
AlertLevel                       METRIC
BIOS Date                        SYSINFO      04/01/2014
BIOS Vendor                      SYSINFO      SeaBIOS
BIOS Version                     SYSINFO      SeaBIOS
BlockedProcesses                 METRIC
BufferMemory                     METRIC
BytesRecv                        COUNTER
BytesSent                        COUNTER
...
```

Filtering And Grouping For Monitoring Reports

A filter can be executed using the `execute` command on a specified field, and applying a filter grouping to it using an operator.

For example, in `report` mode, the already-existing `"Dual cpu"` object by default has the query property:

Example

```
[rima92->monitoring->report]% get dual cpu query
filter processors == 2 group_by cores "processor vendor"
```

With this, the operator == checks for dual CPU cores using a filter grouping on the field `cores`. An existing query can be executed with the `execute -q` option:

Example

```
[rima92->monitoring->report]% execute -q dualcpu
cores processor vendor Hostnames
------- --------------- ---------------------------------------------
2 GenuineIntel nas,rima92-a,rima92-b
```

Similarly, the field `ssh2node` can have the filter grouping `category` applied to it using the operator == for all nodes that are up. This could then be executed.

If the `ssh2node` field output is a `PASS` for nodes that are in a category `gpu`, and `ssh2node` is also a `PASS` for `node001` and `node002`, but is a `FAIL` for `node003` and `node004`, then the report from the query for this, showing the non-empty groupings of hostnames, would be as indicated by the following session:

Example

```
[rima92->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
[rima92->monitoring->report]%
```

Alternatively, the query can be saved and run as follows:

Example

```
[rima92->monitoring->report]% add ssh2node-category
[rima92->monitoring->report*[ssh2node-category*]% set query
```

the editor opens up and the line
`execute filter status == up group_by ssh2node category`
is entered

```
[rima92->monitoring->report*[ssh2node-category*]% commit
[rima92->monitoring->report*[ssh2node-category]% execute
ssh2node category Hostnames
--------- -------- ----------------------
PASS      default  node001,node002
FAIL      default  node003,node004
PASS      gpu      gpu01..gpu20
[rima92->monitoring->report*[ssh2node-category]%
```

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

```
[rima92->monitoring->report]% execute filter status == up group_by ssh2node category --save /tmp/test
[rima92->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

```
[rima92->monitoring->report]% device power status -n ^/tmp/PASS-default.lst
custom ................... [   ON    ] node001
custom ................... [   ON    ] node002
```

8.12.2 Monitoring Report Queries In RIMA View

In RIMA View, report queries can be accessed from the Network Operating Center (NOC) mode (selected by clicking on the hub-and-spoke icon (page 332)). A clickpath to the panel is `Monitoring`→`Report Queries`→`Report Query List`

The RIMA View equivalent to the `cmsh` managmenent of section 8.12.1 is shown in figure 8.36:

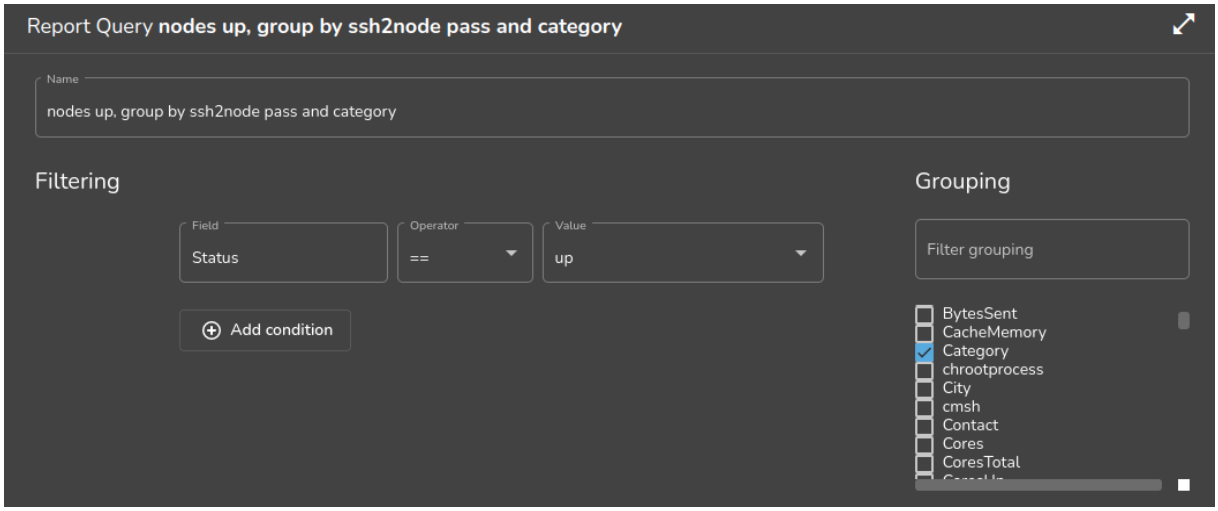


Figure 8.36: RIMA View Monitoring: Report Query For Nodes That Are Up, Grouped By `ssh2node` And `Category`

9

Day-to-day Administration

Updating software packages is an important part of the tasks carried out in daily cluster administration. That topic is substantial enough to have its own chapter (Chapter 7).

Upgrading from NVIDIA Bright Cluster Manager version 9.2 to NVIDIA Base Command Manager version 10 is possible, with care, and is covered in the *Upgrade Manual* at `https://support.brightcomputing.com/upgrade-manuals/10.0/upgrade-manual.pdf`.

This chapter discusses other tasks that may come up in day-to-day administration with RIMA Cluster Manager.

Section 9.1 discusses running shell commands in parallel over the cluster.

Section 9.2 discusses how a cluster administrator can ask the help of the cluster manager support team for guidance with an issue. optimum manner for a cluster administrator.

Section 9.3 discusses how backups can be implemented for the cluster manager.

Section 9.4 discusses revision control for images.

Section 9.5 discusses BIOS configuration with the cluster manager.

Section 9.6 discusses checking hardware matching across the nodes of the cluster.

Section 9.7 discusses Serial Over LAN console access.

Section 9.8 discusses administrative aspects of handling the large amounts of raw monitoring data.

Section 9.9 discusses node replacement.

Section 9.10 discusses using Ansible to configure the cluster via Ansible collections and playbooks.

9.1 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 9.1.1), runs from within the OS shell. That is, `pdsh` executes its commands from within `bash` by default.
- `pexec` (parallel execute, section 9.1.2, 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.

9.1.1 `pdsh` In The OS Shell

Packages Required For `pdsh`

By default, the following packages must be installed from the cluster manager 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 the cluster manager 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@rima92 ~]# cmsh -c "device status"
node001 .................. [   UP   ]
node002 .................. [  DOWN  ]
rima92 ................... [   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@rima92 ~]# pdsh -A hostname
node001: node001
node002: ssh: connect to host node002 port 22: No route to host
pdsh@rima92: node002: ssh exited with exit code 255
rima92: rima92
```

-v: With this option an attempt is made to run the command only on nodes nodes that are in the state `UP`:

Example

```
[root@rima92 ~]# pdsh -A -v hostname
node001: node001
rima92: rima92
```

-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@rima92 ~]# 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@rima92 ~]# pdsh -w node00[1-2] hostname
node001: node001
node002: ssh: connect to host node002 port 22: No route to host
pdsh@rima92: 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@rima92 ~]# 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@rima92 ~]# 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@rima92 ~]# pdsh -A ls /etc/services /etc/yp.conf
rima92: /etc/services
rima92: /etc/yp.conf
node001: /etc/services
node001: ls: cannot access /etc/yp.conf: No such file or directory
pdsh@rima92: node001: ssh exited with exit code 2
node002: /etc/services
node002: /etc/yp.conf
```

With dshbak, with no dshbak options:

Example

```
[root@rima92 ~]# pdsh -A ls /etc/services /etc/yp.conf | dshbak
node001: ls: cannot access /etc/yp.conf: No such file or directory
pdsh@rima92: node001: ssh exited with exit code 2
----------------
rima92
----------------
/etc/services
/etc/yp.conf
----------------
node001
----------------
/etc/services
----------------
node002
----------------
/etc/services
/etc/yp.conf
[root@rima92 ~]#
```

With dshbak, with the -c (coalesce) option:

Example

```
[root@rima92 ~]# pdsh -A ls /etc/services /etc/yp.conf | dshbak -c
node001: ls: cannot access /etc/yp.conf: No such file or directory
pdsh@rima92: node001: ssh exited with exit code 2
----------------
node002,rima92
----------------
/etc/services
/etc/yp.conf
----------------
node001
----------------
/etc/services
[root@rima92 ~]#
```

The dshbak utility is useful for creating human-friendly output in clusters with larger numbers of nodes.

9.1.2 pexec In cmsh

In cmsh, the pexec command is run from device mode:

Example

```
[rima92->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
```

9.1.3 pexec In RIMA View

In RIMA View, pexec is hidden, but executed in a GUI wrapper, using the clickpath 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 RIMA 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 RIMA View is faster in most cases, due to less graphics rendering overhead.

9.1.4 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

```
[rima92->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

```
[rima92->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

```
[rima92->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.

9.1.5 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:

```
[rima92->device]%help <pcommand>
```

9.2 Getting Support With Cluster Manager Issues, And Notifications For Release Updates

The cluster manager is constantly undergoing development. While the result is a robust and well-designed cluster manager, it is possible that the administrator may run into a feature that requires tech-

nical help from RIMA or RIMA's resellers, or a release update. This section describes how to get support for technical issues, and notifications for release updates.

9.2.1 Support Via E-mail

If the reseller from whom the cluster manager was bought offers direct support, then the reseller should be contacted.

Otherwise the primary means of support is e-mail. The e-mail exchanges begin via the RIMA website. The website allows the administrator to submit a support request via a web form, and opens up a trouble ticket.

- A useful indirect starting point that can be used for all technical support is at:

  `https://support.brightcomputing.com`

- Cluster administrators that have a RIMA customer portal account can click on the `Request Support` link in the navigation menu on the left-hand column of the portal page. A direct URL for this is:

  `https://customer.brightcomputing.com/Customer-Portal?p=requestsupport`

- Cluster administrators that do not have a RIMA customer portal account can use the web form at:

  `https://customer.brightcomputing.com/support?p=submitSupportRequest`

The correct, up-to-date, product key (section 4.3.2 of the *Installation Manual*) for the cluster should be entered in the web form to allow tickets to be dealt with smoothly. If the product key for the cluster has been registered (section 4.3 of the *Installation Manual*), then all tickets associated with the key can be viewed after logging in.

When opening up the ticket, it is a good idea to try to use a clear subject header, since that is used as part of a reference tag as the ticket progresses. Also helpful is a good description of the issue. The followup communication for this ticket goes via standard e-mail.

Putting obviously different problems into one ticket results in a slower resolution. Issues that are clearly independent of each other should be submitted as separate tickets. This ensures the quickest processing by the support team.

The document "How To Ask Questions The Smart Way" at `http://www.catb.org/esr/faqs/smart-questions.html` is quite readable, and explains how to submit a good technical query on online forums. While the document is not entirely relevant to the support provided by RIMA, it is true that many of the principles covered there are also useful when sending a query to RIMA when trying to resolve an issue. A read through the document may therefore be helpful before submitting a query to RIMA.

If the issue appears to be a bug, then a bug report should be submitted. It is helpful to include as many details as possible to ensure the development team is able to reproduce the apparent bug. The policy at RIMA is to welcome such reports, to provide feedback to the reporter, and to resolve the problem.

If there is apparently no response from RIMA over e-mail, checking the spam folder is advised. Mail providers have been known to flag mails as spam, even in mid-thread. Users in China should also be aware that mails from RIMA may be deleted by accident by the Golden Shield Project (the "Great Firewall") operated by the Chinese Ministry of Public Security.

As a supplement to e-mail support, RIMA also provides the `cm-diagnose` (section 9.2.2) and the `request-remote-assistance` (section 9.2.3) utilities to help resolve issues.

9.2.2 Reporting Cluster Manager Diagnostics With `cm-diagnose`

The diagnostic utility `cm-diagnose` is run from the head node and gathers data on the system that may be of use in diagnosing issues. To view its options, capabilities, and defaults, it can be run as "`cm-diagnose --help`". For particular issues it may be helpful to change some of the default values to gather data in a more targeted way.

When run without any options, it runs interactively, and allows the administrator to send the resultant diagnostics file to RIMA 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@rima92 ~]# 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

To diagnose any issue, it would help if you could provide a description
about the issue at hand.
Do you want to enter such description? [Y/n]

End input with ctrl-d
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.

If issues are suspected to be in the cmdaemon process, a gdb trace of
that process is useful.
In general such a trace is only needed if RIMA 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
    Processing file contents
    Processing large files and log files
    Collecting process information for CMDaemon
    Executing CMSH commands
    Finished executing CMSH commands

Processing default-image
    Processing commands
    Processing file contents

Creating log file: /root/rima92_609.tar.gz

Cleaning up

Automatically submit diagnostics to
http://support.rima.com/cm-diagnose/ ? [Y/n] y

Uploaded file: rima92_609.tar.gz
Remove log file (/root/rima92_609.tar.gz)?  [y/N] y
[root@rima92 ~]
```

9.2.3 Requesting Remote Support With `request-remote-assistance`

The life-cycle of solving a ticket begins with opening a ticket, and then establishing that both the cluster administrator and the support engineer have a basic grasp of the issue at hand. This is best done via an e-mail exchange.

From this stage onward there are many possible paths. The support engineer may offer a solution, or ask for more details, or may ask for some tests to be run. Most of the time e-mail remains the most efficient way to troubleshoot an issue.

However at times it may be more appropriate for the cluster administrator to allow remote support from the RIMA support engineer in order to resolve the issue. The support engineer may in that case suggest that the `request-remote-assistance` utility be run.

The `request-remote-assistance` utility allows a RIMA engineer to securely tunnel into the cluster, often without a change in firewall or `ssh` settings of the cluster.

With `request-remote-assistance`:

- It must be allowed to access the `www` and `ssh` ports of RIMA's internet servers.
- For some problems, the engineer may wish to power cycle a node. In that case, indicating what node the engineer can power cycle should be added to the option for entering additional information.
- Administrators familiar with `screen` may wish to run it within a screen session and detach it so that they can resume the session from another machine. A very short reminder of the basics of how to run a screen session is: run the `screen` command, then run the `request-remote-assistance` command, then `ctrl-a d` to detach. To resume, run `screen -r`, and `exit` to exit.

The `request-remote-assistance` command itself is run as follows:

Example

```
[root@rima92 ~]# request-remote-assistance

This tool helps securely set up a temporary ssh tunnel to
sandbox.rima.com.

Allow a RIMA engineer ssh access to the cluster? [Y/n]

Enter additional information for RIMA (eg: related
ticket number, problem description)? [Y/n]

End input with ctrl-d
Ticket 1337 - the florbish is grommicking

Thank you.

Added temporary RIMA public key.
```

After the administrator has responded to the `Enter additional information...` entry, and has typed in the `ctrl-d`, the utility tries to establish the connection. The screen clears, and the secure tunnel opens up, displaying the following notice:

```
REMOTE ASSISTANCE REQUEST
########################################################
A connection has been opened to RIMA Support.
Closing this window will terminate the remote assistance
session.
```

```
------------------------------------------------------

Hostname: rima92.NOFQDN
Connected on port: 7000

ctrl-c to terminate this session
```

RIMA support automatically receives an e-mail alert that an engineer can now securely tunnel into the cluster. The session activity is not explicitly visible to the administrator. Whether an engineer is logged in can be viewed with the `w` command, which shows a user running the ssh tunnel, and—if the engineer is logged in—another user session, other than whatever sessions the administrator may be running:

Example

```
[root@rima92 ~]# w
 13:35:00 up 97 days, 17 min, 2 users, load average: 0.28, 0.50, 0.52
USER TTY FROM LOGIN@ IDLE JCPU PCPU WHAT
root pts/0 10.2.37.101 12:24 1:10m 0.14s 0.05s ssh -q -R :7013:127.0.0.1:22\
 remote@sandbox.brightcomputing.com rima92
root pts/1 localhost.locald 12:54 4.00s 0.03s 0.03s -bash
```

When the engineer has ended the session, the administrator may remove the secure tunnel with a `ctrl-c`, and the display then shows:

```
Tunnel to sandbox.rima.com terminated.
Removed temporary RIMA public key.
[root@rima92 ~]#
```

The RIMA engineer is then no longer able to access the cluster.

The preceding tunnel termination output may also show up automatically, without a `ctrl-c` from the administrator, within seconds after the connection session starts. In that case, it typically means that a firewall is blocking access to SSH and WWW to RIMA's internet servers.

9.2.4 Getting Notified About Updates

Updates for the various RIMA Cluster Manager releases continue for some time after the initial release is made public. The updates typically containing bugfixes and improvements, and an administrator may wish to install an update when it becomes publicly available.

An administrator can sign up to be notified about an update by logging into the customer portal at `https://customer.brightcomputing.com/`, and then clicking on `Edit Account` from the navigation menu in the left panel (figure 9.1). An e-mail address can be set in the account details, and the versions for which the administrator would like to receive notifications can be selected by ticking on the appropriate `Notify on releases for:` checkboxes.

After submitting the request, an e-mail is sent to the address when a new release is made public for the selected versions.

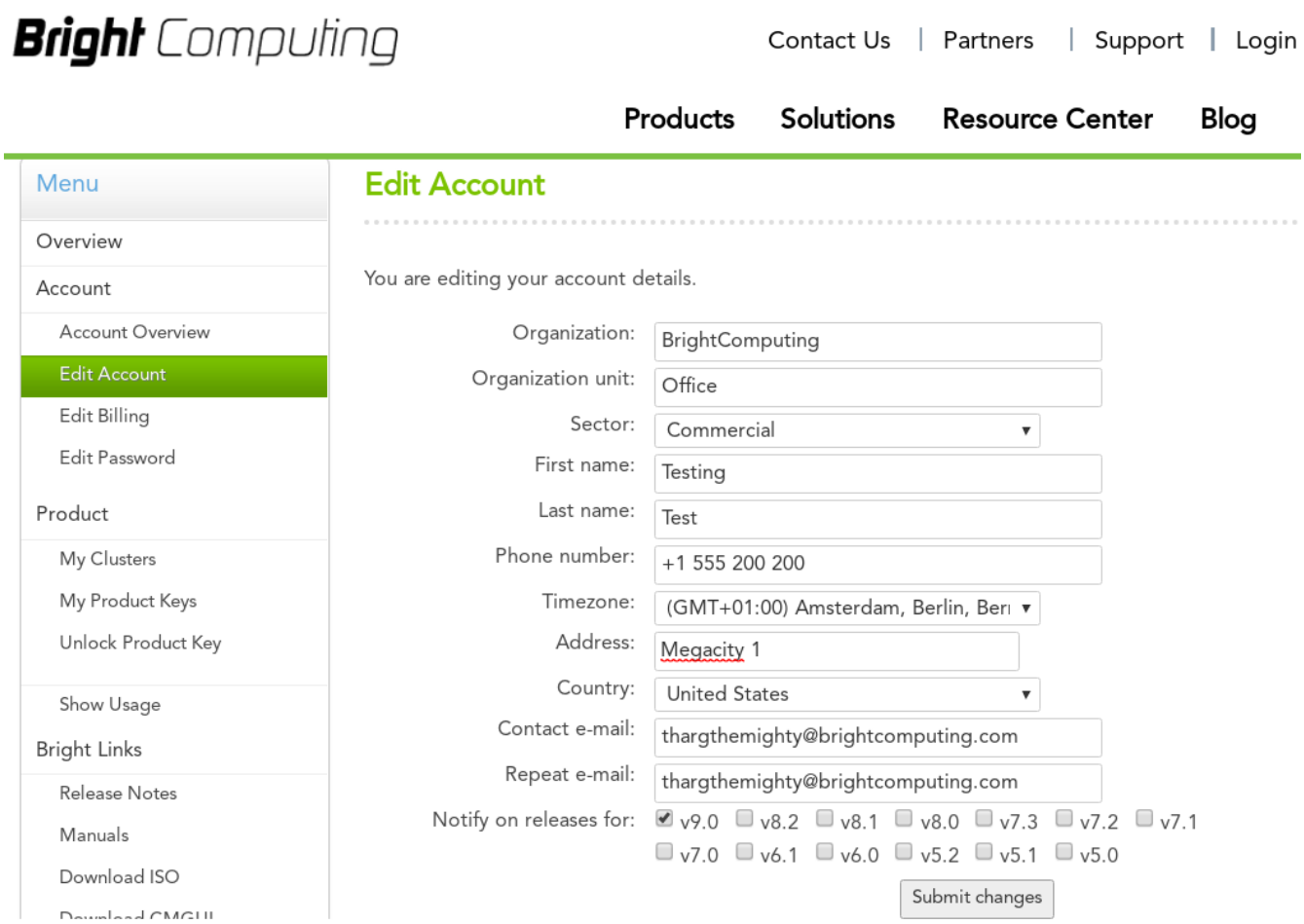


Figure 9.1: Signing up for release notification at the customer portal

9.3 Backups

9.3.1 Cluster Installation Backup

The cluster manager 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 10.4.8.

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. Including the following lines in the Shorewall rules file for the head node allows access via those ports from an IP address of

93.184.216.34 on the external network:

Example

```
ACCEPT  net:93.184.216.34     fw      tcp     9101
ACCEPT  net:93.184.216.34     fw      tcp     9102
ACCEPT  net:93.184.216.34     fw      tcp     9103
```

The Shorewall service should then be restarted 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.

9.3.2 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 also backed up nightly on the cluster filesystem itself ("local rotating backup") for the last 7 days.
- 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 any such by itself.

When there is database corruption, InfoMessages in the `/var/log/cmdaemon` log may mention:

- "`Unexpected eof found`" in association with a table in the database,
- "`can't find file`" when referring to an entire missing table,
- locked tables,
- error numbers from table handlers,
- "`Error while executing`" a command.

A restoration from backup can be carried out to repair the database, as explained in the next section.

Restoring From The Local Backup

If the MySQL InnoDB repair tools do not fix the problem, then, for a failover configuration, the `dbreclone` option (section 10.4.2) 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 additional databases to be cloned. Running the `/cm/local/apps/cmd/scripts/cm-update-mycnf` script then updates `/etc/my.cnf`. 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
service cmd stop #(just to make sure)
mysql -ucmdaemon -p<password> cmdaemon < backup-Tue.sql
```

Running "`service cmd start`" 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 the cluster manager after the time the snapshot was taken are no longer implemented.

Monitoring data values are not kept in a database, but in files (section 9.8).

9.4 Revision Control For Images

RIMA Cluster Manager 7 introduced support for the implementations of Btrfs provided by the distributions. Btrfs makes it possible to carry out revision control for images efficiently.

9.4.1 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 the cluster manager, 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 3.15.4) 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.

9.4.2 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 was made available from Red Hat as a technology preview release in RHEL7, but has been deprecated since RHEL7.4, and is not officially supported by Red Hat in any case. It has been officially removed from RHEL8.

Btrfs is also available from SUSE. SLES12 uses a 3.12 series kernel, which has a fully stable and supported implementation of Btrfs.

While RIMA has not experienced any problems with storing software images on Btrfs using RHEL7, it is highly advisable to keep backups of important software images on a non-Btrfs filesystems when Btrfs is used on these Linux distributions.

An issue with using `cm-clone-install` with Btrfs is described on page 489.

9.4.3 Installing Btrfs To Work With Revision Control Of Images In The Cluster Manager

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@rima92 ~]# 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@rima92 ~]# cd /cm/
[root@rima92 cm]# mv images images2
[root@rima92 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@rima92 cm]# mkfs.btrfs /dev/sdc1
[root@rima92 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@rima92 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@rima92 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@rima92 cm]# mount -t btrfs butter.img images -o loop
[root@rima92 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 the cluster manager 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@rima92 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@rima92 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@rima92 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 the cluster manager to manage image version tracking, since the necessary functionality has been integrated into `cmsh` and RIMA View.

9.4.4 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 9.4.3. 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

```
[rima92->softwareimage]% newrevision default-image "some changes"
[rima92->softwareimage]% newrevision default-image "more changes"
[rima92->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
[rima92->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
[rima92->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

   ```
   [rima92->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

   ```
   [rima92->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

   ```
   [rima92->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 9.4.3.

First, a revision of the image is made to save its initial state:

```
[rima92->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:

```
[rima92->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:

```
[rima92->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`:

```
[rima92->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:

```
[rima92->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.

9.5 BIOS Configuration And Updates

9.5.1 BIOS Configuration Via CMDaemon And Redfish

The cluster manager uses a CMDaemon API to manage the BIOS of nodes in userspace.

In RIMA Cluster Manager version 9.0 and earlier, the implementation varied according to the proprietary vendor implementation.

Starting with RIMA Cluster Manager version 9.1, BIOS management is carried out by CMDaemon making use of 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.

Cluster Manager JSON Configuration Templates In Redfish

The JSON configuration is specified per vendor. In the cluster manager the files are kept under:

```
/cm/local/apps/cm-bios-tools/templates/
```

By default, the cluster manager 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@rima92 ~]# 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@rima92 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"
        },
...
```

cluster manager BIOS Configuration States And Operations Overview

In the cluster manager, 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
RIMA  --------------> CMDaemon database -----------> BIOS (pending) -----------> BIOS (live)
View
```

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 that is stored in the firmware, 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 433). However the `cmsh` or RIMA 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

```
[rima92->device[node002]->biossettings]% status
Parameter                       Configured Pending Live
------------------------------- ---------- ------- ----
Pending errors:
No model defined
```

It can be set with the help of tab-completion:

Example

```
[rima92->device[node002]->biossettings]% set model<tab><tab>
dell_r730  hpe_dl380
[rima92->device[node002]->biossettings]% set model hpe_dl380
[rima92->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

```
[rima92->device[node002]]% biossettings
[rima92->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

```
[rima92->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

   ```
   [rima92->device[node002]->biossettings]% set boot mode<tab><tab>
   legacy bios mode  uefi mode
   [rima92->device[node002]->biossettings]% set boot mode legacy bios mode
   [rima92->device*[node002]->biossettings*]% commit
   [rima92->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

     ```
     [rima92->device[node002]->biossettings]% ..
     [rima92->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

```
[rima92->device[node002]]% bios apply
Node             Result   Output                          Error
---------------- -------- ------------------------------- -------------------------------
node002          good
[rima92->device[node002]]% biossettings
[rima92->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

```
[rima92->device[node002]->biossettings]% ..
[rima92->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

```
[rima92->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 the easiest way for a cluster administrator to manage the BIOS of a cluster via Redfish is usually via the `cmsh` (page 430) or RIMA View front ends.

The `cm-bios-manage` help text is:

```
[root@rima92 ~]# /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@rima92 ~]# /cm/local/apps/cm-bios-tools/python/cm-bios-manage -t
[
  "dell_r730",
  "hpe_dl380"
]
[root@rima92 ~]# /cm/local/apps/cm-bios-tools/python/cm-bios-manage -p hpe_dl380
[
  "test"
]
[root@rima92 ~]# /cm/local/apps/cm-bios-tools/python/cm-bios-manage -P hpe_dl380 test
{
  "test": "xyz"
}
[root@rima92 ~]# /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
    },
...
```

9.5.2 Updating BIOS Versions

There are two ways that the firmware can be updated. A legacy way based on DOS tools (page 434), and a more recent way, based on CMDaemon and Redfish (page 435).

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. 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 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.

Updating A BIOS Via CMDaemon

For systems that support the Redfish protocol, such as HPE iLO5 systems, firmwares can be updated under the `device` mode of `cmsh`, using the `firmware` command. Firmwares updated via Redfish need not be just the PC main system BIOS, but can also be the flashable software of subsystems such as NICs.

The following options are available for the `firmware` command:

- `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`: provides a list of firmware states on specified nodes.
- `upload`: 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. Once it is in that special flash location it can be flashed to where the firmware is actually run.
- `remove`: 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. Possible output values for the state are:
  - `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

The following session shows `node001` getting uploaded and flashed with a firmware, and then having the firmware removed.

Example

```
[rima92->device]% firmware info
Node     Filename     Component        Version  State      Progress Result   Size     Date
-------- ------------ ---------------- -------- ---------- -------- -------- -------- --------------------
rima92 iLO5-2.42    iLO5             2.42     undefined  N/A               8.6MiB   02/28/2022, 11:22:55
rima92 iLO5-2.43    iLO5             2.43     undefined  N/A               8.7MiB   02/28/2022, 11:22:55
rima92 iLO5-2.44    iLO5             2.44     undefined  N/A               8.8MiB   02/28/2022, 11:22:55
rima92 iLO5-2.45    iLO5             2.45     undefined  N/A               8.9MiB   02/28/2022, 11:22:55

[rima92->device]% firmware list -n node001

[rima92->device]% firmware upload iLO5-2.42 -n node001
Node     Result   Output                 Error
-------- -------- ---------------------- --------------------------------
node001  good     uploading: iLO5-2.42

[rima92->device]% firmware list -n node001
Node     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

[rima92->device]% firmware flash iLO5-2.42 -n node001
Node     Result   Output                 Error
-------- -------- ---------------------- --------------------------------
node001  good     flashing: iLO5-2.42

[rima92->device]% firmware status -n node001
Node     Filename     Component        Version  State     Progress Result   Size     Date
-------- ------------ ---------------- -------- --------- -------- -------- -------- --------------------
node001               iLO5             2.42     flashing  35.6%             N/A      02/28/2022, 11:27:27

[rima92->device]% firmware status -n node001
Node     Filename     Component        Version  State      Progress Result   Size     Date
-------- ------------ ---------------- -------- ---------- -------- -------- -------- --------------------
node001               iLO5             2.42     completed  N/A               N/A      02/28/2022, 11:27:27

[rima92->device]% firmware remove iLO5-2.42 -n node001
Node     Result   Output                 Error
-------- -------- ---------------------- --------------------------------
node001  good     removed: iLO5-2.42

[rima92->device]% firmware list -n node001
```

9.6 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@rima92 ~]# cmsh -c "category add newbunch; commit"
[root@rima92 ~]# 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:

```
(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@rima92 ~]# /cm/local/apps/cmd/scripts/healthchecks/node-hardware-pr\
   ofile -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 8.2.10) can then be enabled, and the sampling frequency set as follows:

   ```
   [root@rima92 ~]# cmsh
   [rima92]% monitoring setup use hardware-profile
   [rima92->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:

   ```
   [rima92->monitoring->setup[hardware-profile]]% nodeexecutionfilters
   [rima92->...-profile]->nodeexecutionfilters]% add category filterhwp
   [rima92->...-profile]->nodeexecutionfilters*[filterhwp*]]% set categories newbunch
   [rima92->...-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.

9.7 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 RIMA View via the clickpath:
     `Devices`→`Nodes`→*node*→`Connect`→`ssh`.
2. **Remote shell via CMDaemon.** This is possible if CMDaemon is running on the node and accessible via RIMA View or `cmsh`.
   - In RIMA View, An interactive root shell session can be started up on a node via the clickpath:
     `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 9.7.1)
- the configuration of SOL with RIMA View (section 9.7.2)
- the configuration of SOL with `cmsh` (section 9.7.3)
- the `conman` SOL logger and viewer (section 9.7.4)

9.7.1 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 the cluster manager, or vice versa.

During a factory reset of the node, it is likely that a SOL configuration in the cluster manager will no longer match the configuration on the node BIOS after the node boots. This is because the cluster manager cannot configure these. This is in contrast to the IP address and user authentication settings of the BMC (section 3.7), which the cluster manager is able to configure on reboot.

Background Notes: Node Kernel Configuration

Sections 9.7.2 and 9.7.3 explain how SOL access configuration is set up for the node kernel using RIMA 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 the cluster manager's RIMA 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 the cluster manager:

- 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 9.7.3). 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 the node kernel serial port configuration). Alternatively, COM2, corresponding to ttyS1. 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 the cluster manager, or vice versa.

9.7.2 SOL Console Configuration With RIMA View

In RIMA View, SOL configuration settings can be carried out per image via the clickpath `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`.

9.7.3 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@rima92 ~]# cmsh
[rima92]% softwareimage use default-image

[rima92->softwareimage[default-image]]% set enablesol yes
[rima92->softwareimage*[default-image*]]% commit
```

The SOL settings for a particular image can be seen with the `show` command:

```
[rima92->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.

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@rima92 ~]# cmsh
[rima92]% device use node001
[rima92->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:
```

9.7.4 The `conman` Serial Console Logger And Viewer

In the cluster manager, 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 RIMA View, or if the `enablesol` 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 9.2).

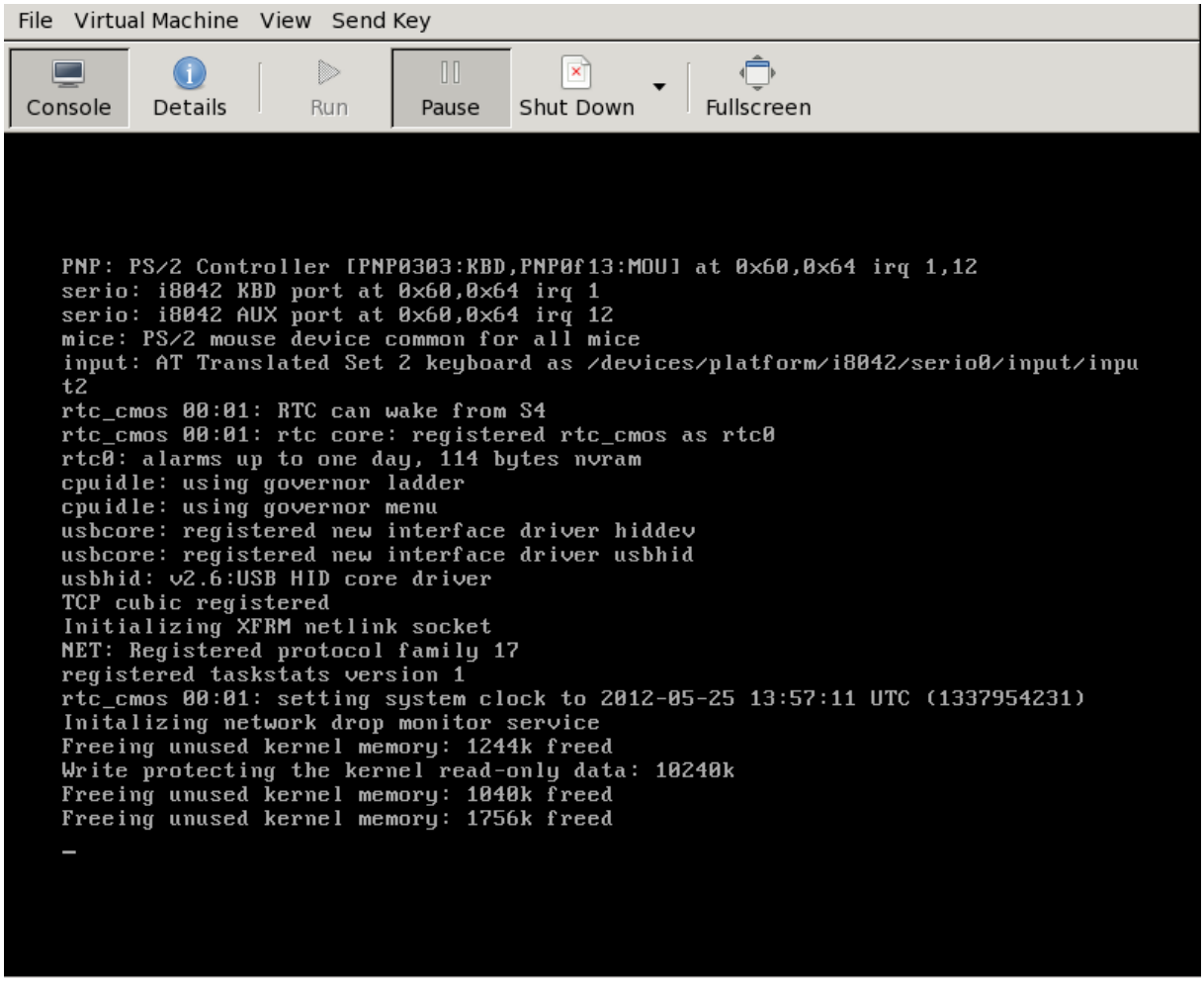


Figure 9.2: Physical Console Freeze During SOL Access

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 9.3) during the freeze of the physical console display.

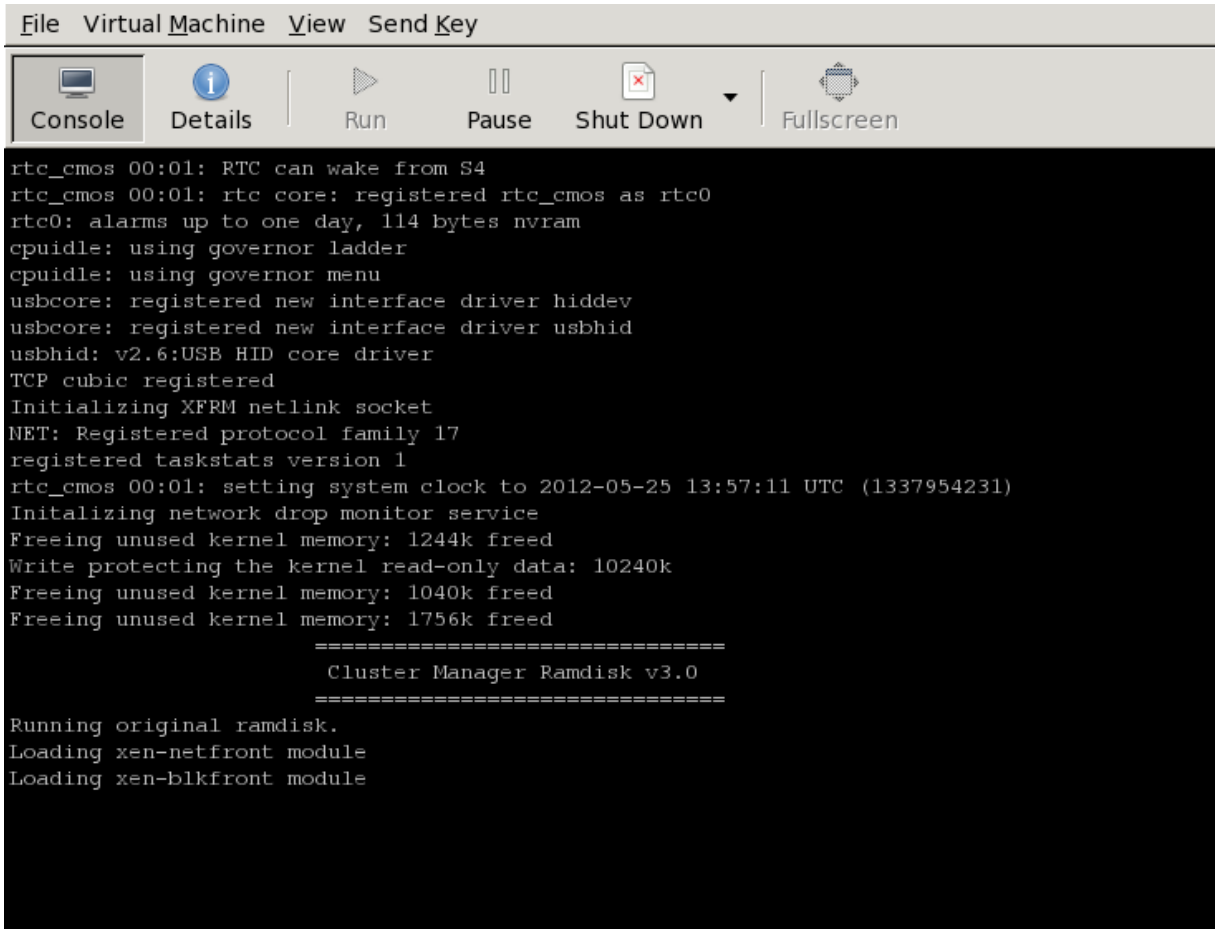


Figure 9.3: Remote Console Continues During SOL Access During Physical Console Freeze

Finally, just before login is displayed, the physical console once more (figure 9.4) starts to display what is on the remote console (figure 9.5).

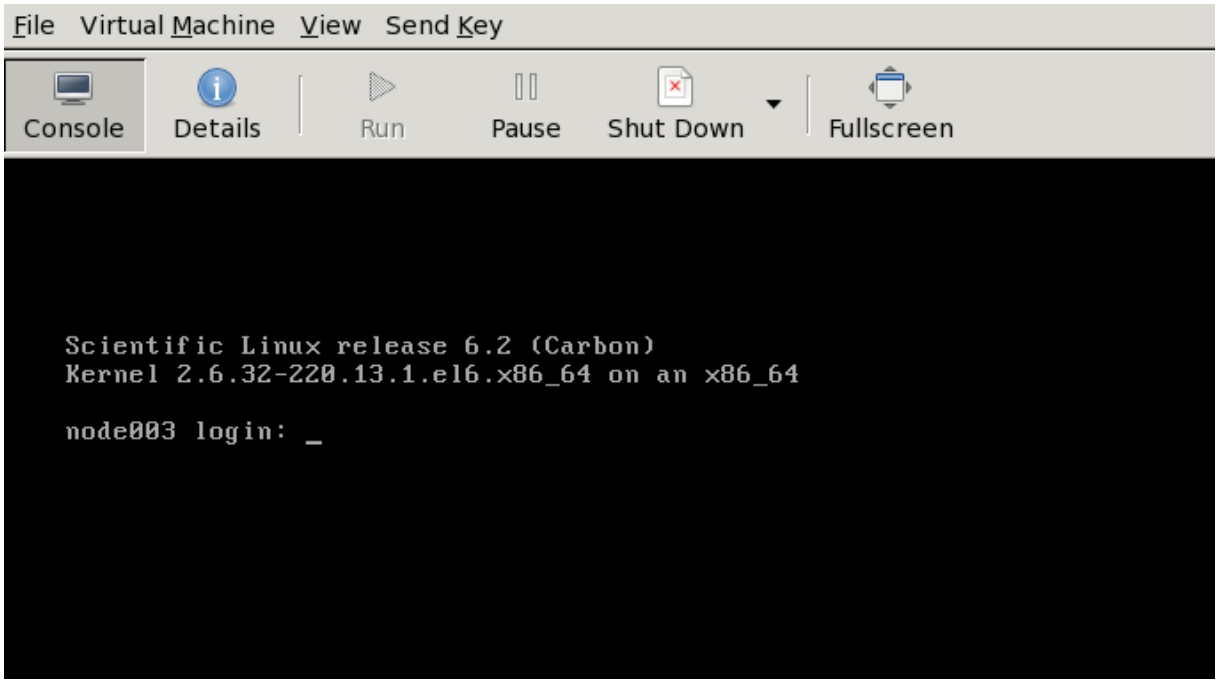


Figure 9.4: Physical Console Resumes After Freeze During SOL Access

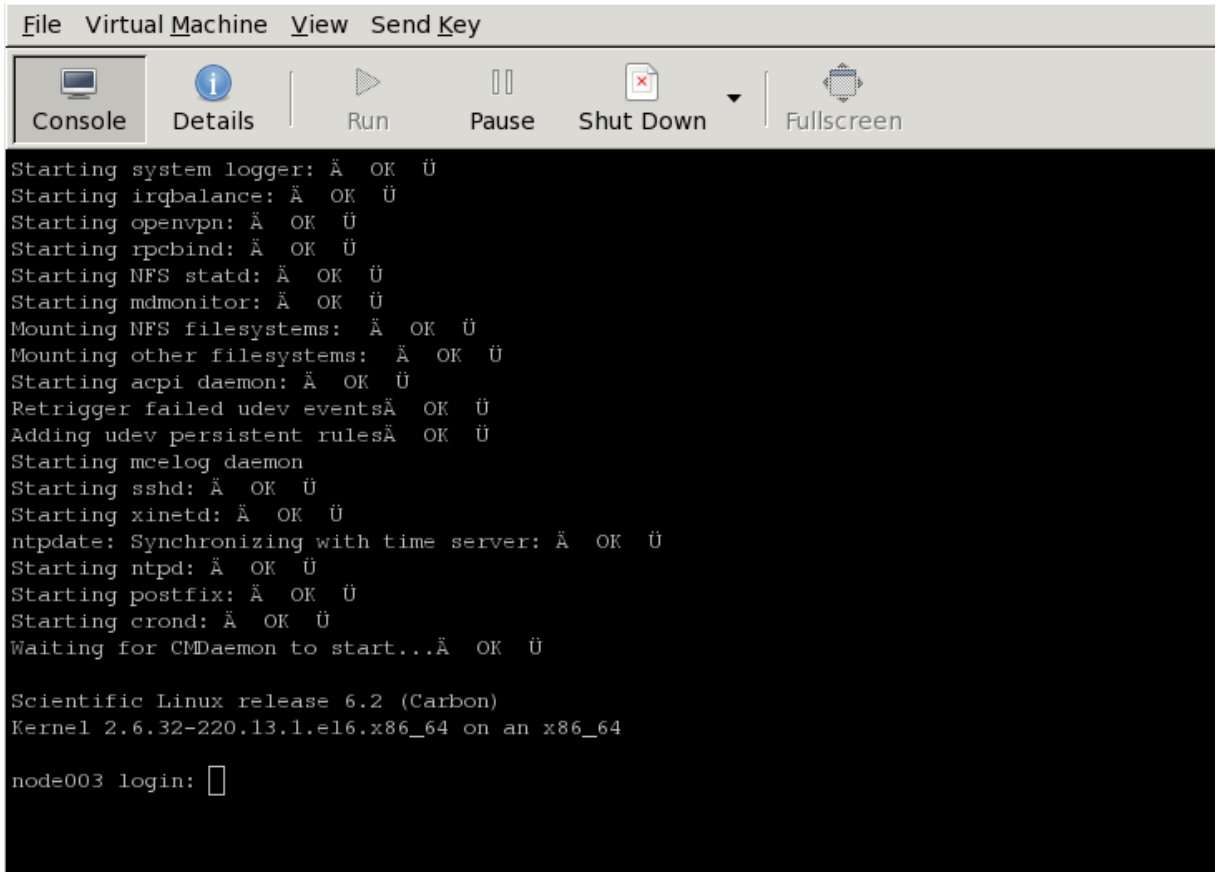


Figure 9.5: Remote Console End Display After Boot

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*>.

9.8 Managing Raw Monitoring Data

From RIMA Cluster 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 the cluster manager was thoroughly rewritten for this change.

9.8.1 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

```
[rima92->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%.

9.8.2 Estimating The Required Size Of The Storage Device

The final size of the monitoring directory can be estimated with the cluster manager script, `cm-bright-monitoring-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@rima92 ~]# /cm/local/apps/cmd/scripts/monitoring/cm-bright-monitoring-usage.py
Number of used entities:      6
Number of used measurables:   231
Number of measurables:        231
Number of data producers:     95
Number of consolidators:      2

Current monitoring directory: /var/spool/cmd/monitoring
Monitoring directory size:    1.024 GB
Maximal directory size:       1.125 GB
```

9.8.3 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 530) is given the new directory as its value.

3. CMDaemon is stopped (`service cmd stop`).

4. The `/var/spool/cmd/monitoring/` directory is moved to the new directory.

5. CMDaemon is restarted (`service cmd start`).

9.8.4 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 8.4.1) to smaller, but still non-zero values. After re-initializing the monitoring data, so that existing data is removed, the values reported by the `cm-bright-monitoring-usage.py` script (section 9.8.2) then show the new storage estimates for the monitoring data.

9.8.5 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 service cmd stop`

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 are cleared 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@rima92 ~]# 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@rima92 ~]# 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@rima92 ~]# 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]> exit
   ```
   repeat on other head node

4. On both head nodes, CMDaemon can then be restarted with:
   `service cmd start`

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 service cmd start`

9.9 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@rima92 ~]# cmsh
[rima92]% device use node0031
if new mac address is known, then:
[rima92->device[node031]]% set mac <new mac address>
else if new mac address is not known:
[rima92->device[node031]]% clear mac
the changed setting in either case must be committed:
[rima92->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 206) and the `Nodes Identification` resource of RIMA View (page 210).

9.10 Ansible And RIMA Cluster Manager

This section describes using Ansible with the cluster manager. Using Ansible to install RIMA Cluster Manager is described in section 3.4 of the *Installation Manual*.

9.10.1 An Overview Of Ansible

Ansible is a popular automated configuration management software.

The cluster manager 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 RIMA Ansible Collections

In particular, the web interface at `https://galaxy.ansible.com/brightcomputing` shows the updated list of the cluster manager Ansible collections.

From version 9.1 onward of RIMA, the RIMA Ansible collection naming scheme has been changed so that the name now indicates the RIMA 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 RIMA Ansible collection for a 9.2 cluster, the following command can now be run:

```
[root@rima92 ~]# ansible-galaxy collection install rima.rcm92
```

9.10.2 A Simple Playbook Example

In this section, a playbook from the cluster manager collection is run.

Preparations

To start with, Python is loaded, and Ansible installed:

```
[root@rima92 ~]# module load python3
[root@rima92 ~]# pip install ansible
```

Running A Simple Playbook

The directory `/cm/local/examples/cmd/ansible` has several cluster manager Ansible playbook examples.

The Ansible playbook to add a user can be run. The playbook is simply:

```
[root@rima92 ~]# cat /cm/local/examples/cmd/ansible/add-user.yaml
---
- hosts: all
  gather_facts: false

  tasks:
    - name: create test-user
      rima.rcm92.user:
        name: test-user
        password: test-user-password
        profile: readonly
```

The latest RIMA 9.2-compatible version of the RIMA Ansible collection is at `https://galaxy.ansible.com/brightcomputing/bcm92`. It can be installed with the `ansible-galaxy` tool from the `galaxy.ansible.com` repository directly with:

```
[root@rima92 ~]# ansible-galaxy collection install rima.rcm92
```

Documentation For The Cluster Manager Collection

The `rima.rcm92` documentation for modules can be explored using `ansible-doc` in the usual way, using the namespace. For example, for the `user` module in the `rima.rcm92` namespace, this would be (output truncated):

```
[root@rima92 ~]# ansible-doc rima.rcm92.user
> BRIGHTCOMPUTING.BCM92.USER
> (/root/.ansible/collections/ansible_collections/brightcomputing/bcm92/plugins/modules/user.py)

        User

ADDED IN: version 9.2.0 of rima.rcm92
```

```
  * 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

- cloudJob
        Create a certificate with the cloudjob profile for cmjob
        [Default: False]
        type: bool

- email
...
```

The list of modules in the `rima.rcm92` collection can be viewed with `ansible-doc -l rima.rcm92`.

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@rima92 ~]# 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@rima92 ~]# cmsh -c "user list"
Name (key)       ID (key)         Primary group    Secondary groups
---------------- ---------------- ---------------- ----------------
cmsupport        1000             cmsupport
test-user        1001             test-user
```

9.10.3 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 the cluster manager infrastructure.

This section (9.10.3) and the next (9.10.4) aim to provide this familiarity. They should be a guide for users when they go about defining their own cluster manager infrastructure with Ansible, as well as a model for how to carry out Ansible tasks for the cluster manager.

The example session in this section (section 9.10.3) 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 the cluster manager can be displayed and used to carry out the cloning as shown in the following session:

```
[root@rima92 ~]# cat /cm/local/examples/cmd/ansible/clone-software-image.yaml
---
- hosts: all
  gather_facts: false
  tasks:
    - name: clone a software image
      rima.rcm92.software_image:
        name: cloned-image
        cloneFrom: default-image
        path: /cm/images/cloned-image

[root@rima92 ~] 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 `rima.rcm92.category` module:

```
[root@rima92 ~]# cat clonedefaultcat.yaml
- hosts: all
  gather_facts: false

  tasks:
    - name: clone category
      rima.rcm92.category:
        name: democategory
        cloneFrom: default
[root@rima92 ~] 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@rima92 ~]# cat setimageincat.yaml
- hosts: all
  gather_facts: false

  tasks:
    - name: set image in category
      rima.rcm92.category:
        name: democategory
        softwareImageProxy:
          parentSoftwareImage: cloned-image
[root@rima92 ~] 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@rima92 ~]# cat setcatofnodes.yaml
- hosts: all
  gather_facts: false

  tasks:
    - name: list all nodes
      rima.rcm92.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
      rima.rcm92.physical_node:
        hostname: "{{ item }}"
        mac: "{{ all_nodes[item].mac }}"
        category: democategory
      loop:
        - node001
        - node002
[root@rima92 ~] ansible-playbook -i localhost, setcatofnodes.yaml
```

Without using Ansible, and using cmsh directly instead, the preceding placement could be carried out with:

```
[root@rima92 ~] for i in {001..002}
do cmsh -c "device use node$i; set category democategory; commit"
done
```

9.10.4 A More Complicated Playbook Example: Creating An Edge Site And Related Properties

This section provides a more complicated RIMA 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 455.

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
      rima.rcm92.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
      rima.rcm92.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
  rima.rcm92.software_image:
    name: my-software-image
    path: /cm/images/my-software-image
    cloneFrom: default-image

- name: create edge director category
  rima.rcm92.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
  rima.rcm92.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
      rima.rcm92.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
      rima.rcm92.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)
  rima.rcm92.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)
  rima.rcm92.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 rima.rcm92.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
  rima.rcm92.software_image:
    name: my-software-image
    path: /cm/images/my-software-image
```

```
    cloneFrom: default-image   #(1)

- name: create edge director category
  rima.rcm92.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
  rima.rcm92.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
  rima.rcm92.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
  rima.rcm92.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.

10

High Availability

10.0 Introduction

10.0.1 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 availablity 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.

10.0.2 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 the cluster manager 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 10.5). HA for regular nodes is a more recent feature in the cluster manager, 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 2.13 of the *Cloudbursting Manual*.

10.0.3 High Availability Usually Uses Shared Storage

HA is typically configured using shared storage (section 10.1.5), such as from an NFS service, which typically provides the `/home` directory on the active (section 10.1.1) head, and on the regular nodes.

10.0.4 Organization Of This Chapter

The remaining sections of this chapter are organized as follows:

- **HA On Head Nodes**
  - Section 10.1 describes the concepts behind HA, keeping the cluster manager configuration in mind.
  - Section 10.2 describes the normal user-interactive way in which the cluster manager implementation of a failover setup is configured.
  - Section 10.3 describes the implementation of the cluster manager failover setup in a less user-interactive way, which avoids using the ncurses dialogs of section 10.2
  - Section 10.4 describes how HA is managed with the cluster manager after it has been set up.
- **HA On Regular Nodes**
  - Section 10.5 describes the concepts behind HA for regular nodes, and how to configure HA for them.
- **HA And Workload Manager Jobs**
  - Section 10.6 describes the support for workload manager job continuation during HA failover.

10.1 HA Concepts

10.1.1 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 RIMA Cluster Manager should not be confused with the definition for primary that is used by workload managers such as Slurm and PBS Professional when a failover mechanism is configured by the workload manager (section **??**).

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.

10.1.2 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.

10.1.3 Services In The Cluster Manager 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 the cluster manager 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)
- **NTP**
- **DNS**
- **Workload Management**: For each of the Slurm, PBS, LSF services, one server is active on one head node, while the other server is a passive standby on the other head node

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 RIMA View is used, then the clickpath `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 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 SGE/GE server (`sgemaster`)

10.1.4 Failover Network Topology

A two-head failover network layout is illustrated in figure 10.1.

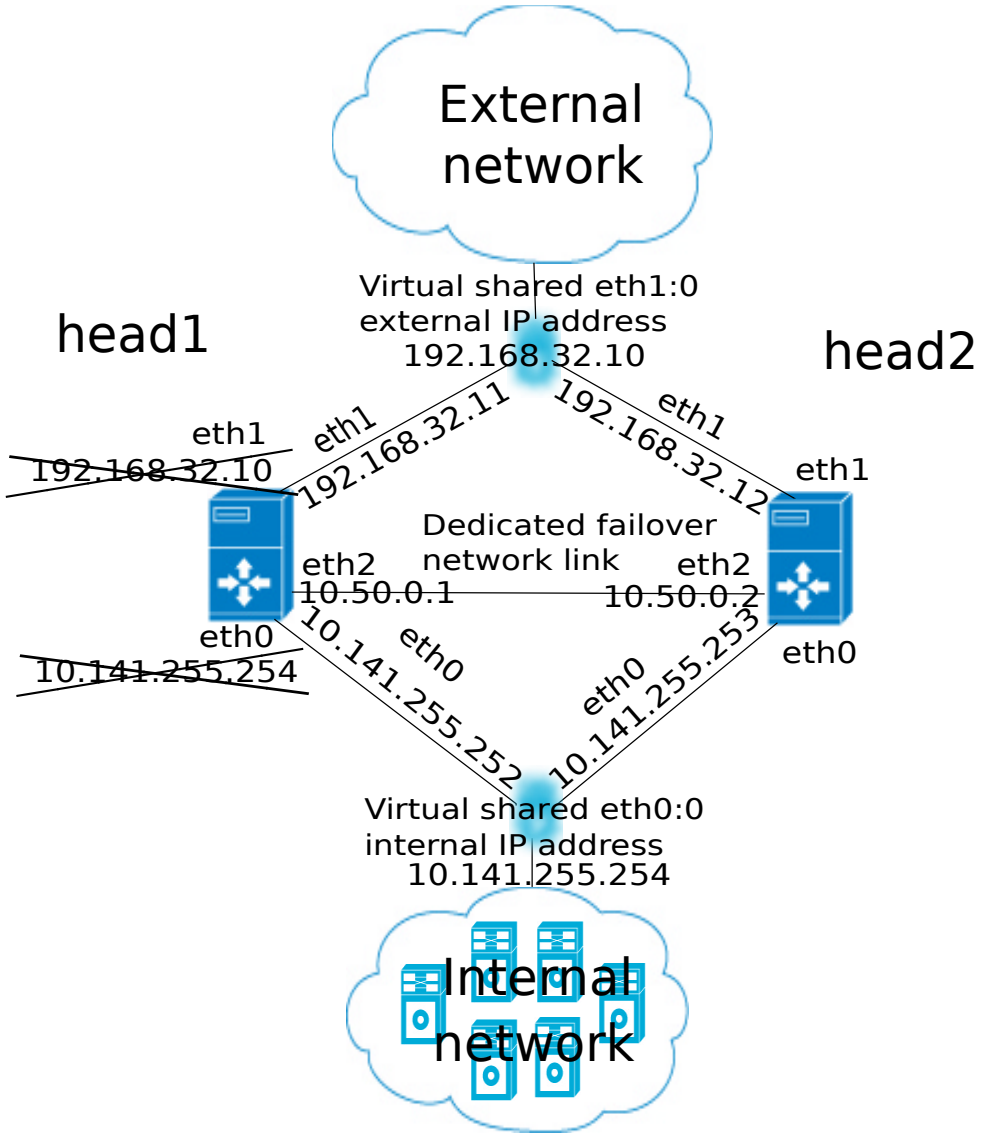


Figure 10.1: High Availability: Two-Head Failover Network Topology

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 10.1. 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.

10.1.5 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.

Although the cluster manager gives the administrator full flexibility on how shared storage is implemented between two head nodes, there are generally two types of storage used: NAS, DAS.

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.

DAS

In a Direct Attached Storage (DAS) setup, both head nodes share access to a block device, often accessed through a SCSI interface. This could be a disk-array that is connected to both head nodes, or it could be a block device that is exported by a corporate SAN infrastructure.

Although the block device is visible and can physically be accessed simultaneously on both head nodes, the filesystem that is used on the block device is typically not suited for simultaneous access. Simultaneous access to a filesystem from two head nodes must therefore be avoided because it generally

leads to filesystem corruption. Only special purpose parallel filesystems such as GPFS and Lustre are capable of being accessed by two head nodes simultaneously.

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 RIMA View (section 10.4.6).

10.1.6 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 10.1.7. 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 the same as a power-off operation (section 10.2.4).

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 the cluster manager 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.

10.1.7 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 10.4.6):

```
[root@rima92 ~]# cmsh
[rima92]% partition failover base
[rima92->partition[base]->failover]% set disableautomaticfailover yes
[rima92->partition*[base*]->failover*]% commit
```

With RIMA View it is carried out via the clickpath `Cluster`→`Partition[base]`→`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 10.4.2), then the failover sequence must also be initiated manually by the administrator.

10.1.8 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 the cluster manager.

10.1.9 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 cluster manager 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 the cluster manager. 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.

10.2 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 the cluster manager 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
```

HA-enabled clusters display two MAC addresses in the output. Single-headed clusters show only one.

If an HA license is not present, it should be obtained from the cluster manager reseller, and then be activated 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 54 of the *Installation Manual*) on RIMA View session logout. This means:

- If LDAP is managed by the cluster manager, then on logout, new user certificates are generated, and a new RIMA View login session picks up the new certificates automatically.
- For LDAPs other than that of the cluster manager, 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 RIMA 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 10.2.1): the configuration parameters are set for the shared interface and for the secondary head node that is about to be installed.
2. **Cloning** (section 10.2.2): the secondary head node is installed by cloning it from the primary head node.
3. **Shared Storage Setup** (section 10.2.3): the method for shared storage is chosen and set up.

An optional extra stage is:

4. **Automated Failover Setup** (section 10.2.4): Power control to allow automated failover is set up.

10.2.1 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 10.2.1-10.2.3, and possibly section 10.2.4) 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 10.2).
4. `Configure` is selected from the `Setup` 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 obtained from the RIMA Cluster Manager reseller, and activated (section 4.3 of the *Installation Manual*).

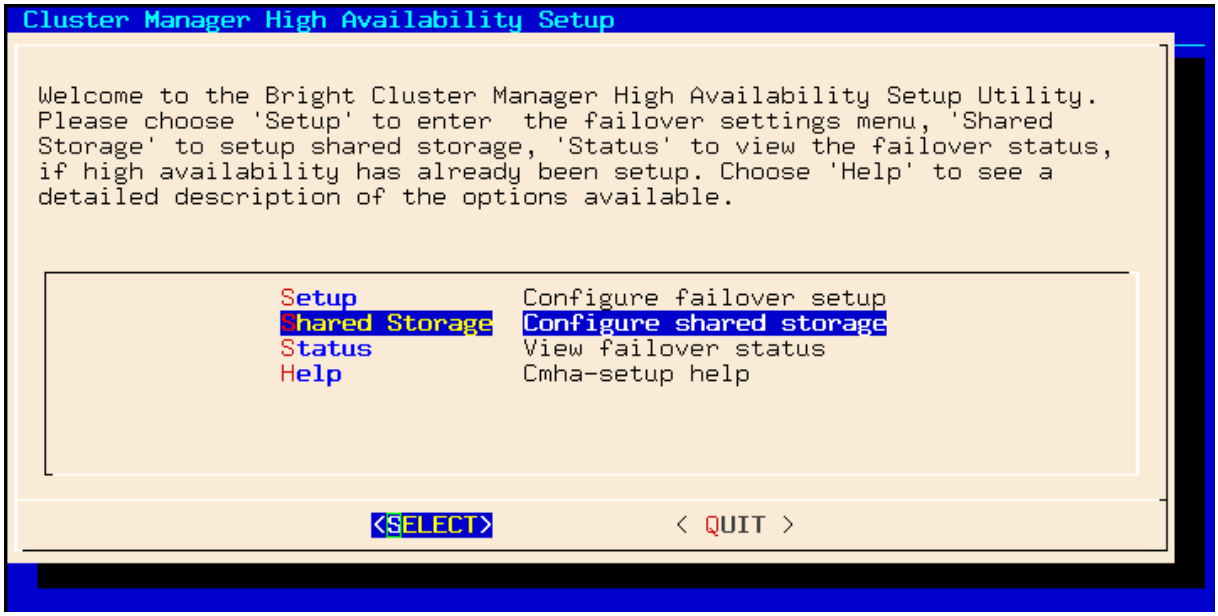


Figure 10.2: `cmha-setup` Main menu

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. The procedure continues further after the password is entered.
16. Setup progress for the planned configuration is displayed (figure 10.3).

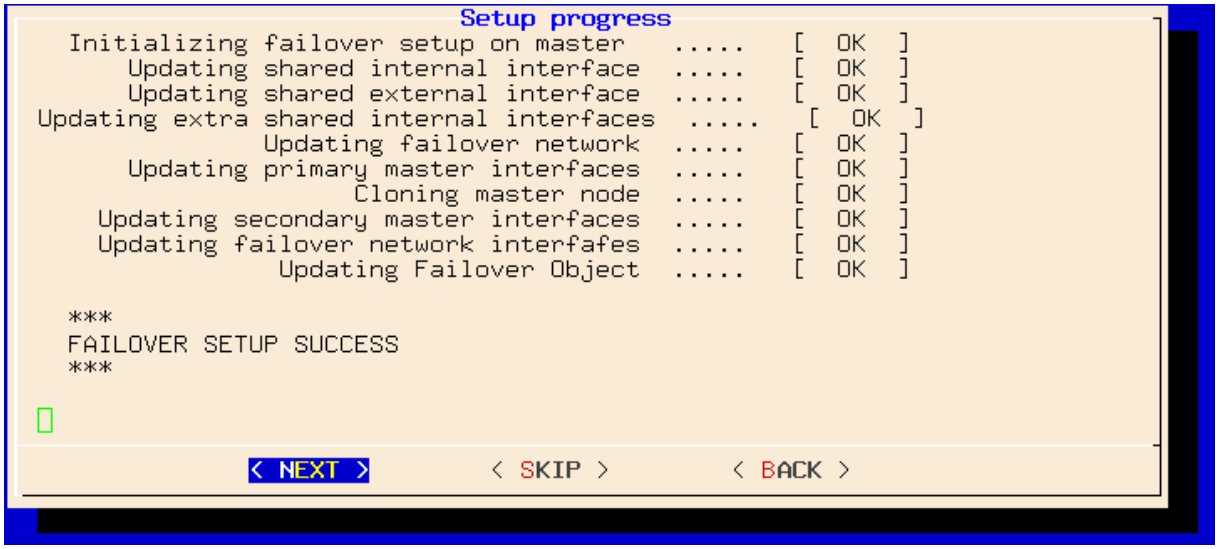


Figure 10.3: cmha-setup Setup Progress For Planned Configuration

17. Instructions on what to run on the secondary to clone it from the primary are displayed (figure 10.4).

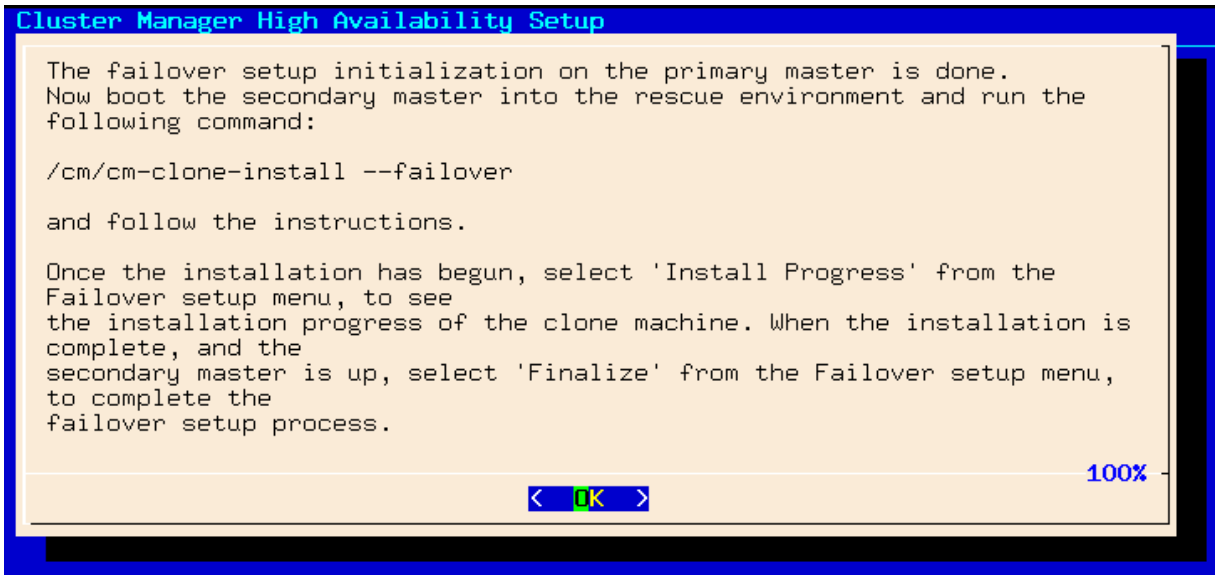


Figure 10.4: cmha-setup Instructions To Run On Secondary For Cloning

10.2.2 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 the cluster manager the word "image" is normally used for the software that can be placed on regular nodes. So, the cluster manager 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 10.4.8.

After the preparation has been done by configuring parameters as outlined in section 10.2.1, 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 10.4.8). In that case the active head node can be either the primary or secondary.

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 10.5).

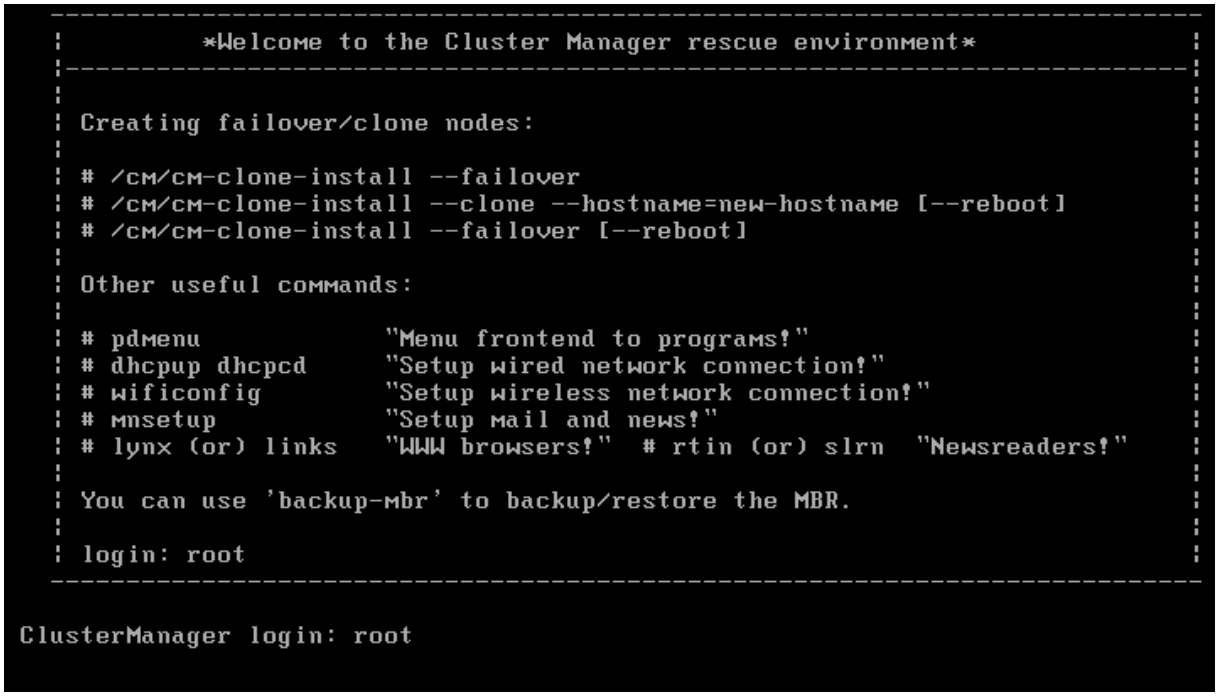


Figure 10.5: Login Screen After Booting Passive Into Rescue Mode From Active

4. The following command is executed (figure 10.6) on the node that is to become a failover clone:
   `/cm/cm-clone-install --failover`

   When doing a re-clone as in section 10.4.8, instead of a failover clone, then it is the `--clone` option that is used instead of the `--failover` option.

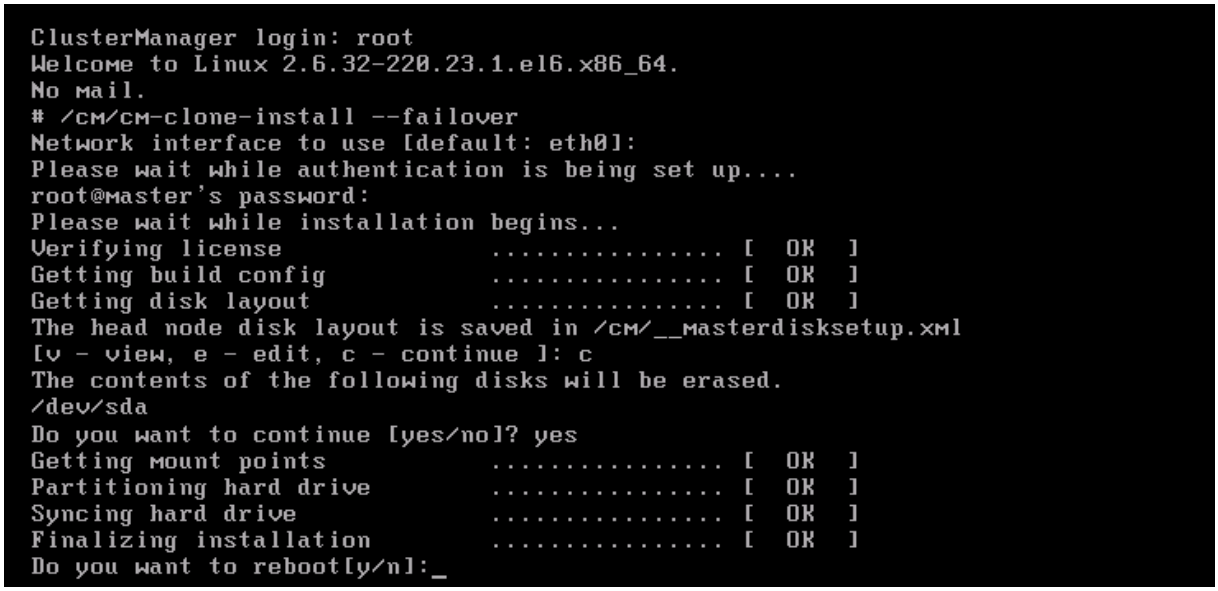


Figure 10.6: Cloning The Passive From The Active Via A Rescue Mode Session

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 entering the MySQL password, the progress of the `Finalize` procedure is 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 10.2.3) 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@rima92 ~]# cmha status
Node Status: running in active master mode

Failover status:
rima92* -> master2
  failoverping [  OK  ]
  mysql        [  OK  ]
  ping         [  OK  ]
  status       [  OK  ]
master2 -> rima92*
  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.

10.2.3 Shared Storage Setup

After cloning the head node (section 10.2.2), the last basic stage of creating an HA setup is setting up shared storage. The available shared storage forms are NAS and DAS.

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 10.1.5. 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 466).
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.

DAS

A prerequisite to the DAS configuration steps that follow is that the partitions exist on the DAS device that is to be used for shared storage. These should be created manually if required, before running cmha-setup.

1. In the cmha-setup main menu, the "Shared Storage" option is selected.
2. DAS is selected.
3. The filesystems that are to be shared over the cluster are selected by the administrator. The filesystems that are shared are typically /cm/shared and /home, but this is not mandatory.
4. The filesystems that are to be shared are assigned DAS partitions by the administrator. For example, the administrator may specify these as /dev/sdc1 for /home and /dev/sdd3 for /cm/shared.
5. The administrator can choose to create a filesystem on the proposed DAS device.
   - Creating the filesystem on the device means any existing filesystems and files on the device are wiped during creation.
   - Otherwise, the existing filesystem and files on the DAS device remain.
6. A filesystem type is set from a choice of ext3, ext4, xfs.
7. A summary screen is presented showing the proposed changes.
8. After filesystems have been created, the current contents of the shared directories are copied onto the shared filesystems and the shared filesystems are mounted over the old non-shared filesystems.
9. The administrator should check that the partitions are visible from both head nodes using, for example, the fdisk -l command on the device. If the partitions on the DAS are created or modified, or appear only after the passive head is running due to a hardware-related reason after the passive head is powered on, then the kernel on the passive head may not have reread the partition table. A power cycle of the head nodes is recommended if the partitions are not seen properly.

10.2.4 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 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 10.4). Database administration with the dbreclone command (section 10.4) 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 RIMA 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 10.4.2). 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.

10.3 Running cmha-setup Without ncurses, Using An XML Specification

10.3.1 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.

10.3.2 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>] ]
```

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 10.2.1. 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 carries clones the passive node from the active, as described in steps 1 to 10 of section 10.2.2.
- -f|--finalize: After the passive node is cloned as described in steps 1 to 10 of section 10.2.2, 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 10.2.2.
  - -r|--finalizereboot: makes the passive reboot after the finalize step completes.
- -s|--sharedstorage <*type*>: specifies the shared storage <*type*> out of a choice of nas, das.
- -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.

There is little attempt at validation with the express mode, and invalid entries can cause the command to hang.

10.3.3 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@rima92 ~]# cd /cm/local/apps/cluster-tools/ha/conf
[root@rima92 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 10.2.1.

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 10.2.2.

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@rima92 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 master2  .....    [  OK  ]
                         Cloning database  .....    [  OK  ]
                    Update DB permissions  .....    [  OK  ]
  Checking for dedicated failover network  .....    [  OK  ]
A reboot has been issued on master2
```

The preceding corresponds to steps 11 to 13 of section 10.2.2.

Adding Storage:

Continuing with the session on the active, setting up a shared storage could be done with:

```
[root@rima92 conf]# cmha-setup -x -c failoverconf.xml -s nas
```

The preceding corresponds to carrying out the NAS procedure of section 10.2.3.

10.4 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.

10.4.1 Changing An Existing Failover Configuration

Changing an existing failover configuration is usually done most simply by running through the HA setup procedure of section 10.2 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 10.2.1.

10.4.2 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 cluster manager 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 RIMA 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, the cluster manager 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 10.2.4).

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 10.1.7), 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 223).

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*. 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 467) 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 RIMA support.

Example

```
[root@rima92 ~]# cmha status
Node Status: running in active master mode

Failover status:
rima92* -> head2
  failoverping [  OK  ]
  mysql        [  OK  ]
  ping         [  OK  ]
  status       [  OK  ]
head2 -> rima92*
  failoverping [  OK  ]
  mysql        [FAILED] (11)
  ping         [  OK  ]
  status       [  OK  ]
[root@rima92 ~]# cmha dbreclone head2
Proceeding will cause the contents of the cmdaemon state database on he\
ad2 to be resynchronized from this node (i.e. rima92 -> 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@rima92 ~]# cmha status
Node Status: running in active master mode

Failover status:
rima92* -> head2
  failoverping [  OK  ]
  mysql        [  OK  ]
  ping         [  OK  ]
  status       [  OK  ]
head2 -> rima92*
  failoverping [  OK  ]
  mysql        [  OK  ]
  ping         [  OK  ]
  status       [  OK  ]
```

10.4.3 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.

10.4.4 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 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 10.4.2).
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 succesful, the passive head becomes the new active head.

10.4.5 Keeping Head Nodes In Sync

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 189).

  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 RIMA 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 7.2)
  - 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 10.2, 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 10.1.3 should also have their packages updated on both head nodes.

For the services that run simultaneously on the head nodes, such as CMDaemon, DHCP, LDAP, MySQL, NTP and DNS, their packages should be updated on both head nodes at about the same time. A suggested procedure is to stop the service on both nodes around the same time, update the service and ensure that it is restarted.

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 10.1.3.

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 10.4.2), 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.

10.4.6 High Availability Parameters

There are several HA-related parameters that can be tuned. Accessing these via RIMA View is described in section 10.4.6. 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 10.1.7 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.

10.4.7 Viewing Failover Via RIMA View

Accessing `cmsh` HA Parameters (`partition failover base`) Via RIMA View
The RIMA View equivalents of the `cmsh` HA parameters in section 10.4.6 are accessed from the clickpath
`Cluster`→`Partition[base]`→`Settings`
→`Failover`

10.4.8 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 10.2.2. Instead of running the `cm-clone-install --failover` command as in that section, the following command can be run:

```
[root@rima92 ~]# /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 481) before carrying out tasks with `cmsh` or RIMA 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 10.2.2).

- `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 RIMA View or cmsh options, and are maintained by CMDaemon. On the other hand, the exclude lists introduced in this section (10.4.8):

- excludelistfailover
- excludelistnormal

are not RIMA 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 9.4.1) 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.

10.5 HA For Regular Nodes And Edge Director Nodes

HA for regular nodes is available from RIMA Cluster Manager version 7.0 onward. HA for edge director nodes follows a similar design to HA for regular nodes, and is available from RIMA Cluster Manager version 9.2 onward.

10.5.1 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 10.0.1).

10.5.2 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 10.1.7).
Warn time and dead time parameters can be set (section 10.4.6).
Mount and unmount scripts (page 466).
Pre- and post- failover scripts (section 10.4.6).
Disabling or enabling automatic failover (section 10.1.7).
A virtual shared IP address that is presented as the virtual node that is always up (section 10.1.4).

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 10.2).	Installed by administrator using a procedure similar to section 10.5.3.	Installed by administrator using `cm-edge-setup` (section 2.1.1 of the *Edge Manual*)
Configurable within `failover` submode of `partition` mode (section 10.4.6).	Configurable within `failovergroups` submode of `partition` mode (section 10.5.3).	Configurable within `failovergroups` submode of `partition` mode (section 10.5.3). 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 10.1.6). 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*).

10.5.3 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

```
[rima92->partition[base]->failovergroups]% status
No active failover groups
[rima92->partition[base]->failovergroups]% add cupsgroup
[rima92->partition*[base*]->failovergroups*[cupsgroup*]]% list
Name (key)               Nodes
------------------------ ------------------------
cupsgroup
```

By default, CUPS is provided in the standard image, in a stopped state. In RIMA View a failover group can be added via the clickpath `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, the cluster manager ensures that one of the nodes in the failover group becomes designated as the active one in the group (some text elided):

Example

```
[rima92->...[cupsgroup*]]% set nodes node001..node002
[rima92->...[cupsgroup*]]% commit
[rima92->...[cupsgroup]]%
...Failover group cupsgroup, make node001 become active
...Failover group cupsgroup, failover complete. node001 became active
[rima92->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

```
[rima92->partition[base]->failovergroups[cupsgroup]]% device
[rima92->device]% foreach -n node001..node002 (services; add cups; \
set runif active; set autostart yes; set monitored yes)
[rima92->device]% commit
Successfully committed 2 Devices
[rima92->device]%
Mon Apr  7 08:45:54 2014 [notice] node001: Service cups was started
```

The `runif` options are described in section 3.11.1.

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

```
[rima92->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

```
[rima92->partition[base]->failovergroups]% use cupsgroup
[rima92->...]->failovergroups[cupsgroup]]% makeactive node002
node002 becoming active ...
[rima92->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 10.4.2):

Example

```
[root@rima92 ~]# 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

```
[rima92->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

```
[rima92->device]% foreach -n node001..node002 (interfaces; add alias \
bootif:0 ; set ip 10.141.240.1; set startif active; set network internalnet)
[rima92->device*]% commit
Successfully committed 2 Devices
[rima92->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

```
[rima92->...interfaces*[BOOTIF:0*]]% set additionalhostnames cups
[rima92->...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 3.15.4
- by saving and using a new image with the modifications, as suggested for greater modifications in section 3.15.2, page 158.

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 10.2.4.

Example

```
ssh node001
echo c > /proc/sysrq-trigger
~.
```

A passive node then takes over.

10.5.4 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 492):

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.

10.6 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 the workload manager and HA storage pairings indicated by the following table:

Table 10.6: HA Support For Jobs Vs Shared Filesystems

WLM	DAS	NAS
Slurm	Y	Y
Grid Engine	Y	Y
PBS Professional	N	Y
LSF	N	Y

As the table shows, PBS Professional, and LSF are not able to support HA for jobs on DAS filesystems. This is because they require daemons to run on the passive and active nodes at the same time, in order to provide access to `/cm/shared` at all times. During failover the daemons normally cannot run on the node that has failed, which means that the DAS storage types cannot provide access to `/cm/shared` during this time. Job continuity cannot therefore be supported for these workload manager and storage type combinations.

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 cluster manager prejob health check configuration.

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), in the software images (section A.2), and on the regular nodes (sections A.4 and A.4). These files should not be confused with configuration files that are merely installed (section A.5).

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 CMDeamon- 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 CMDeamon On Head Nodes

If the `filewriteinfo` command is run for the head node, then it displays the files that have been modified by CMDaemon on the head node. Running the `filewriteinfo` command for the head node might display the following on a fresh installation:

```
[rima92->device]% filewriteinfo rima92
Hostname  Path                                        Timestamp                 Actor  Frozen
--------- ------------------------------------------- ------------------------- ------ --------
rima92  /cm/images/default-image/etc/securetty     Mon Jun 27 09:46:14 2022  cmd    no
rima92  /etc/chrony.conf                           Mon Jun 27 09:46:00 2022  cmd    no
rima92  /etc/dhcp/dhclient.conf                    Mon Jun 27 09:46:14 2022  cmd    no
rima92  /etc/postfix/canonical                     Mon Jun 27 09:46:13 2022  cmd    no
rima92  /etc/postfix/generic                       Mon Jun 27 09:46:13 2022  cmd    no
rima92  /etc/postfix/main.cf                       Mon Jun 27 09:46:13 2022  cmd    no
rima92  /etc/resolv.conf                           Mon Jun 27 09:46:14 2022  cmd    no
rima92  /etc/shorewall.staging/interfaces          Mon Jun 27 09:46:18 2022  cmd    no
rima92  /etc/shorewall.staging/netmap              Mon Jun 27 09:46:18 2022  cmd    no
rima92  /etc/shorewall.staging/policy              Mon Jun 27 09:46:18 2022  cmd    no
rima92  /etc/shorewall.staging/snat                Mon Jun 27 09:46:18 2022  cmd    no
rima92  /etc/shorewall.staging/zones               Mon Jun 27 09:46:18 2022  cmd    no
rima92  /etc/sysconfig/network-scripts/ifcfg-eth1  Mon Jun 27 09:46:14 2022  cmd    no
rima92  /tftpboot/images/default-image//boot       Mon Jun 27 09:47:02 2022  cmd    no
```

```
rima92  /tftpboot/images/default-image/initrd      Mon Jun 27 09:47:02 2022  cmd    no
rima92  /tftpboot/images/default-image/vmlinuz     Mon Jun 27 09:47:02 2022  cmd    no
rima92  /tftpboot/mtu.conf                         Mon Jun 27 09:46:14 2022  cmd    no
[rima92->device]%
```

The `filewriteinfo` command is covered in more detail in section A.3.

The more important head node files that are managed by CMDaemon are listed here for a plain installation on the various distributions.

Files generated or modified automatically on the head node by CMDaemon In RHEL and derivatives, Ubuntu 18.04, Ubuntu 20.04, SLES12, SLES15

File	Part	Comment
/cm/local/apps/openldap/etc/slapd.conf	Section	CMDaemon modifies this when edge, cloud or HA are activated
/cm/local/apps/<PBS>/var/cm/cm-pbs.conf	Section	<PBS> can be one of openpbs pbspro
/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 and derivatives only
/etc/dhcpd.conf	Entire file	
/etc/dhcp/dhclient.conf	Section	RHEL7,8 and derivatives, Ubuntu 18.04, Ubuntu 20.04
/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/fstab	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. For zone additions use /etc/named.conf.include[1]. For options additions, use /etc/named.conf.global.options.include.
/etc/ntp.conf	Section	RHEL7 and derivatives only
/etc/ntp/step-tickers	Section	RHEL7 and derivatives only
/etc/postfix/canonical	Section	
/etc/postfix/generic	Section	
/etc/postfix/main.cf	Section	
/etc/resolv.conf	Section	

...continues

...continued

File	Part	Comment
/etc/shorewall6/interfaces	Section	
/etc/shorewall6/policy	Section	
/etc/shorewall6/rules	Section	
/etc/shorewall6/snat	Section	
/etc/shorewall6/zones	Section	
/etc/shorewall/interfaces	Section	
/etc/shorewall/netmap	Section	
/etc/shorewall/policy	Section	
/etc/shorewall/rules	Section	
/etc/shorewall/snat	Section	
/etc/shorewall/zones	Section	
/etc/snmp/snmptrapd.conf	Section	
/etc/sysconfig/bmccfg	Entire file	BMC configuration
/etc/sysconfig/clock	Section	
/etc/sysconfig/dhcpd	Entire file	
/etc/sysconfig/network-scripts/ifcfg-*	Section	
/tftpboot/mtu.conf	Entire file	RIMA configuration
/tftpboot/pxelinux.cfg/category.default	Entire file	RIMA configuration
/var/lib/named/*.zone[1]	Entire file	For SLES distributions only. For custom additions use /var/lib/named/*.zone.include
/var/named/*.zone[1]	Entire file	For non-SLES 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

A.2 System Configuration Files Created Or Modified By CMDaemon In Software Images:

The following is a list of CMDaemon-managed system configuration files in the software images, that is, under /cm/images/<*image*>:

System configuration files created or modified by CMDaemon in software images in RHEL7/8 and derivatives, SLES12, SLES15, Ubuntu18.04, Ubuntu 20.04

File	Part	Comment
/etc/aliases	Section	
/etc/hosts	Section	
/etc/mkinitrd_cm.conf	Section	Not for Ubuntu
/etc/initramfs-tools/modules	Section	Only for Ubuntu
/etc/modprobe.d/rima-cmdaemon.conf	Section	
/etc/postfix/main.cf	Section	
/etc/rsyslog.conf	Section	
/etc/securetty	Section	
/etc/shorewall/rules	Section	
/etc/shorewall/snat	Section	
/etc/shorewall6/rules	Section	
/etc/shorewall6/snat	Section	
/etc/sysconfig/clock	Section	

A.3 System Configuration Files Created Or Modified By CMDaemon 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

```
[rima92->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.

The following construction can be used to view files changed on all nodes:

```
[rima92->device]% foreach * (filewriteinfo)
```

A.3.1 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 45).
- `--field`: The columns headers of the output of the `filewriteinfo` command, such as `Actor`, are not keys for the `--field` option.

Example

```
[rima92->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 34) is run.

The `--field` option can be used with the `mac` field, for example, as follows (some output elided):

Example

```
[rima92->device]% get node002 mac
FA:16:3E:54:BB:A1
[rima92->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` on page 501.

- `--sort`: This sorts the `filewriteinfo` output according to column headers (`Actor`, `Timestamp` and so on).
- `--path`: This can be used to specify a file.

Example

```
[rima92->device]% filewriteinfo  --path /etc/chrony.conf --sort actor
Hostname         Path              Timestamp                 Actor           Frozen
---------------- ----------------- ------------------------- --------------- ------------
rima92         /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.4 Files Created On Regular Nodes By The Node-Installer

The list of files on the 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. For a default installation in RHEL7/8 and derivatives, the list is as shown in the following table:

System configuration files created or modified on regular nodes by the node-installer in RHEL7/8 and derivatives

File	Part	Comment
/etc/sysconfig/network-scripts/ifcfg-*	Entire file	not ifcfg-log
/etc/sysconfig/network	Section	
/etc/resolv.conf	Entire file	
/etc/hosts	Section	
/etc/postfix/main.cf	Section	
/etc/chrony.conf	Entire file	RHEL8 and derivatives only
/etc/ntp.conf	Entire file	RHEL7 and derivatives only
/etc/ntp/step-tickers	Entire file	RHEL7 and derivatives only
/etc/hostname	Section	
/cm/local/apps/cmd/etc/cert.pem	Section	
/cm/local/apps/cmd/etc/cert.key	Section	
/cm/local/apps/openldap/etc/certs/ldap.pem	Section	
/cm/local/apps/openldap/etc/certs/ldap.key	Section	
/var/spool/cmd/disks.xml	Section	
/cm/local/apps/cmd/etc/cluster.pem	Section	
/var/log/rsyncd.log	Section	
/var/log/node-installer	Section	
/var/log/modified-by-node-installer.log	Section	

The files that are modified by node-installer can be seen using the filewriteinfo command for the node.

A.5 Files Not Generated, But Installed In RHEL And Derivatives

This appendix (Appendix A) is mainly about generated configuration files. This section (A.5) 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 the cluster manager 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 cluster manager 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@rima92 ~]# repoquery -l $(repoquery -a | grep -F _cm9.2) | grep nsswitch.conf$
```

The inner repoquery displays a list of all the packages. By grepping for the cluster manager version string, for example _cm9.2 for RIMA Cluster Manager 9.2, 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 the cluster manager 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 cluster manager 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 the cluster manager packages are updated. During such a cluster manager update, a notification is displayed that new configuration files are available.

Inverting the cluster manager version string match displays the files not provided by the cluster manager. These are normally the files provided by the distribution:

```
[root@rima92 ~]# repoquery -l $(repoquery -a | grep -F -v _cm9.2) | grep nsswitch.conf$
/usr/share/factory/etc/nsswitch.conf
/usr/share/factory/etc/nsswitch.conf
/usr/share/authselect/default/minimal/nsswitch.conf
/usr/share/authselect/default/nis/nsswitch.conf
/usr/share/authselect/default/sssd/nsswitch.conf
/usr/share/authselect/default/winbind/nsswitch.conf
/var/lib/authselect/nsswitch.conf
/usr/share/authselect/default/minimal/nsswitch.conf
/usr/share/authselect/default/nis/nsswitch.conf
/usr/share/authselect/default/sssd/nsswitch.conf
/usr/share/authselect/default/winbind/nsswitch.conf
/var/lib/authselect/nsswitch.conf
/usr/share/rear/skel/default/etc/nsswitch.conf
/etc/nsswitch.conf
/etc/nsswitch.conf
/usr/share/doc/yp-tools/nsswitch.conf
```

Which package provides the file in the cluster manager 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 the cluster manager 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, A.2, and A.4 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 cluster manager package, queries similar to the following can be run:

Example

```
root@head:~# dpkg-query -W -f='${binary:Package} ${Version}\n' $(dpkg -S "nsswitch.conf" | cut -f1 -d:) |\
 grep cm9.1
cm-config-ldap-client-master 9.1-131-cm9.1
root@head:~# dpkg-query -W -f='${binary:Package} ${Version}\n' $(dpkg -S "nsswitch.conf" | cut -f1 -d:) |\
 grep -v cm9.1
libc-bin 2.27-3ubuntu1
manpages 4.15-1
```

What are the differences between the cluster manager 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-2.17
~# rpm2cpio glibc-2.17-292.el7.x86_64.rpm | cpio -idmv
~# yumdownloader cm-config-ldap-client-9.2
~# rpm2cpio  cm-config-ldap-client-9.2-114_cm9.2.noarch.rpm
~# 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 the cluster manager 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[1]. 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.

When building the cluster manager 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.

The cluster manager should however mark all critical files as configuration files in the cluster manager 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:

[1]This freezing should not be confused with the `FrozenFile` directive (Appendix C), where the file or section of a file is being maintained by CMDaemon, and where freezing the file prevents CMDaemon from maintaining it.

- 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 7.3.2), thereby avoiding the overwriting of that file by the excluded package.

A request to change the package build flag may be sent to RIMA if the preceding suggested options are unworkable.

B

RIMA Public Key


```
-----BEGIN PGP PUBLIC KEY BLOCK-----
Version: GnuPG v1.4.0 (GNU/Linux)

mQGiBEqtYegRBADStdQjn1XxbYorXbFGncF2IcMFiNA7hamARt4w7hjtwZoKGHbC
zSLsQTmgZO+FZs+tXcZa5OLjGwhpxT6qhCe8Y7zIh2vwKrKlaAVKj2PUU28vKj1p
2W/OIiG/HKLtahLiCkOL3ahPOevJHh8B7elClrZOTKTBB6qIUbC5vHtjiwCgydm3
THLJsKnwk4qZetluTupldOEEANCzJ1nZxZzN6ZAMkIBrct8GivWClT1nBG4UwjHd
EDcGlREJxpg/OhpEP8TY1e0YUKRWvMqSVChPzkLUTIsd/O4RGTwOPGCo6Q3TLXpM
RVoonYPR1tRymPNZyW8VJeTUEnOkdlCaqZykp1sRb3jFAiJIRCmBRc854i/jRXmo
foTPBACJQyoEH9Qfe3VcqR6+vR2tX9lPvkxS7A5AnJIRs3Sv6yM4oV+7k/HrfYKt
fyl6widtEbQ187Os4x3NYXmmne7lz1nGxBfAxzPG9rtjRSXyVxc+KGVd6gKeCV6d
o7kS/LJHRiOLb5G4NZRFy5CGqg64liJwp/f2J4uyRbC8b+/LQbQ7QnJpZ2hOIENv
bXB1dGluZyBEZXZlbG9wbWVudCBUZWFtIDxkZXZAYnJpZ2hOY29tcHVOaW5nLmNv
bT6IXgQTEQIAHgUCSq1h6AIbAwYLCQgHAwIDFQIDAxYCAQIeAQIXgAAKCRDvaS9m
+k3m0JO0AKC0GLTZiqoCQ6TRWW2ijjITEQ8CXACgg3o4oVbrG67VFzHUntcA0YTE
DXW5AgOESq1h6xAIAMJiaZI/0EqnrhSfiMsMT3sxz3mZkrQQL82Fob7s+S7nnMl8
A8btPzLlK8NzZytCglrIwPCYG6vfza/nkvyKEPh/f2it941bh7qiu4rBLqr+kGx3
zepSMRqIzW5FpIrUgDZOL9J+tWSSUtPW0YQ5jBBJrgJ8LQy9dK2RhAOLuHfbOSVB
JLIwNKxafkhMRwDoUNS4BiZKWyPFu47vd8fM67IPT1nMl0iCOR/QBn29MYuWnBcw
61344pd/IjOu3gM6YBqmRRU6yBeVi0TxxbYYnWcts6tEGAlTjHUOQ7gxVp4RDia2
jLVtbee8H464wxkkC3SSkng216RaBBAoaAykhzcAAwUH/iG4WsJHFw3+CRhUqy51
jnmb1FTFO8KQXI8J1PXMOh6vv0PtP5rw5D5V2cyVe2i4ez9Y8XMVfcbf60lptKyY
bRUjQq+9SNjt12ESU67YyLstSN68ach9Af03PoSZIKkiNwfA0+VBILv2Mhn7xd74
5LOM/eJ7lHSpeJA2Rzs6szc2340b/VxGfGWjogaK3NElSYOzQo+/kOVMdMWsQm/8
Ras19IA9P5jlSbcZQlHlPjndS4x4XQ8P41ATczsIDyWhsJC51rTuw9/QO7fqvvPn
xsRz1pFmiiN7I4JLjwOnAlXexn4EaeVa7Eb+uTjvxJZNdShs7Td74OmlF7RKFccI
wLuISQQYEQIACQUCSq1h6wIbDAAKCRDvaS9m+k3m0C/oAJsHMmKrLPhjCdZyHbB1
e19+5JABUwCfUOPoawBNOHzDnfr3MLaTgCwjsEE=
=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:

  ```
  service cmd restart
  ```

- 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 523.

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. Shorewall 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 RIMA 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. The firewall port settings value does however change if using the `cm-cmd-ports` utility (page 65 of the *Installation Manual*) instead.

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.

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 cluster manager 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 cluster manager 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 cluster manager 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: `DBHost = 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 8.10.3) 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 8.10.3) 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 G.

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 RIMA View.

The connection runs over CMDaemon, which is running over SSL, which means that between RIMA View and the device, the connection is encrypted.

The directive does not affect `cmsh`'s `rshell`, `rconsole` `telnet`, and `ssh` commands.

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.

CMDaemonAudit directive

Syntax: `CMDaemonAudit = yes|no`
Default: `CMDaemonAudit = no`

When the CMDaemonAudit directive is set to yes, 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) defines the services that CMDaemon provides for that profile user. More than one profile can be set as a comma-separated list. Out of the profiles that are available on a newly-installed system: node, admin, cmhealth, and readonly; only the profile node is enabled by default. New profiles can also be created via the profile mode of cmsh or via the clickpath Identity Management→Profiles→ of RIMA View, thus making it possible to disable auditing for arbitrary groups of CMDaemon services.

EventLogger directive

Syntax: EventLogger = true|false
Default: EventLogger = false

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 519).

Whether events are logged in files or not, events are cached and accessible using cmsh or RIMA View. The number of events cached by CMDaemon is determined by the parameter MaxEventHistory (page 519).

EventLoggerFile directive

Syntax: EventLoggerFile = *filename*
Default: EventLogger = "/var/spool/cmd/events.log"

The EventLogger directive sets where the events seen in the event viewer (section 8.10.1) are logged.

MaxEventHistory directive

Syntax: AdvancedConfig = {"MaxEventHistory=*number*", ...}
Default: MaxEventHistory=8192
MaxEventHistory is a parameter of the AdvancedConfig (page 528) 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 519) 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: *none*

The `MaximalSearchDomains` directive is a parameter of the `GlobalConfig` (page 529) directive.

By default, the number of names that can be set as search domains used by the cluster has a maximum limit of 6. This is a hardcoded limit imposed by the Linux operating system in older versions.

More recent versions of glibc (glibc > 2.17-222.el7 in RHEL7) no longer set a limit. However using more than 6 search domains currently requires the use of the GlobalConfig directive, `MaximalSearchDomains`. For example, to set 30 domains, the directive setting would be: `GlobalConfig = { "MaximalSearchDomains=30" }`

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 the cluster manager has no MAC addresses associated with the node. The directive `LockDownDhcpd` is deprecated because its globality affects clients on all networks managed by the cluster manager, 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 RIMA 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 RIMA 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 5.8.7.

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 528) directive.

The parameter takes a full file path to a script that can be used for identification with a BMC (section 3.7.4).

BMCIdentifyScriptTimeout directive

Syntax: `AdvancedConfig = {"BMCIdentifyScriptTimeout=`*number from 1 to 360*`", ...}`
Default: `60`
`BMCIdentifyScriptTimeout` is a parameter of the `AdvancedConfig` (page 528) 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 528) 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 528) 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 cluster manager 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: FreezeChangesToPBSPro directive FreezeChangesToUGEConfig directive 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:

- PBSPro
- GE

- 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.

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 RIMA View (section 5.6.3). After the update, CMDaemon should be restarted within that category of nodes:

Example

```
[root@rima92 ~]# pdsh -v -g category=default service cmd restart
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 7.7) 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 309.

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.

The Necessity Of A FrozenFile Directive

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 RIMA support.
- `2` allows violation of the requirements, without validation warnings. This value should never be set except under instructions from RIMA 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. the cluster manager 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 528) 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 cluster manager Knowledge Base at `http://kb.rima.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 528) directive.

`ResolveMasterToExternalName` can only be used if `ResolveToExternalName` (page 526) 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 528) directive.

`ResolveMasterToExternalDomainName` can only be used if `ResolveToExternalName` (page 526) 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 J.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@rima92 ~]# 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@rima92 ~]# 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@rima92 ~]# cm-manipulate-advanced-config.py -r key8
Updated: /cm/local/apps/cmd/etc/cmd.conf
[root@rima92 ~]# 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@rima92 ~]# 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 520).

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 8 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 338) 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 528) directive.

Its value determines the path of the directory in which monitoring data is saved (section 9.8).

MaxServiceFailureCount directive

Syntax: AdvancedConfig = {"MaxServiceFailureCount=*number*", ...}
Default: Implicit value: "MaxServiceFailureCount=10"

MaxServiceFailureCount is a parameter of the AdvancedConfig (page 528) directive.

Its value determines the number of times a service failure event is logged (page 128). 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 528) 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 530), 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

```
# service cmd restart
```

A more refined approach that avoids a complete CMDaemon restart would be to execute a `reset` (page 128) on the `fhgfs-client` from within CMDaemon, as follows:

```
[rima92->category[default]->services[fhgfs-client]]% reset fhgfs-client
Successfully reset service fhgfs-client on: node001,node002
[rima92->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 528) 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.

CookieCooldownTime directive

Syntax: `AdvancedConfig = {"CookieCooldownTime=`*number from 60 to 86400*`", ...}`
Default: 900

`CookieCooldownTime` is a parameter of the `AdvancedConfig` (page 528) directive.

It defines the number of seconds until the RIMA View connection to CMDaemon times out, if there is no user activity at the RIMA View client side.

DHCPMaxleaseTime directive

Syntax: `AdvancedConfig = {"DHCPMaxleaseTime=`*number*`", ...}`
Default: *client default*

`DHCPMaxleaseTime` is a parameter of the `AdvancedConfig` (page 528) 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.

SlurmConfigureMIGs directive

Syntax: `AdvancedConfig = {"SlurmConfigureMIGs=0|1", ...}`
Default: `1`

`SlurmConfigureMIGs` is a parameter of the `AdvancedConfig` (page 528) directive. If set to `0`, then Slurm does not configure the MIGs for the Slurm workload manager.

SlurmDisableAccountingParsing directive

Syntax: `AdvancedConfig = {"SlurmDisableAccountingParsing=0|1", ...}`
Default: `0`

`SlurmDisableAccountingParsing` is a parameter of the `AdvancedConfig` (page 528) directive. If set to `1`, it disables collection of accounting information for the Slurm workload manager.

SlurmHardwareAutoDetection directive

Syntax: `AdvancedConfig = {"SlurmHardwareAutoDetection=0|1", ...}`
Default: `0`

`SlurmHardwareAutoDetection` is a parameter of the `AdvancedConfig` (page 528) directive. If set to `1`, it enables automated hardware detection for Slurm for the following parameters (page **??**):

- `corespersocket`
- `threadspercore`
- `boards`
- `socketsperboard`
- `sockets`
- `realmemory`

The autodetected values are added to the `NodeName` line in `slurm.conf`.

SlurmDrainReasonPolicy directive

Syntax: `AdvancedConfig = {"SlurmDrainReasonPolicy=<`*policy*`>", ...}`
Default: `SlurmDrainReasonPolicy=replace`

The `SlurmDrainReasonPolicy` directive is a parameter of the `AdvancedConfig` (page 528) directive.

The `drain` command (section **??**) can be used by a user to drain a node, or a monitoring trigger (section 8.4.5) can make CMDaemon carry out a drain action if, for example, a health check fails. Outside of CMDaemon control, Slurm's own `scontrol` command (`man(1) scontrol`) can be used.

A drain reason can be specified. By default, the policy is that if a node is already drained, then a second drain command with a new drain reason replaces the old drain reason.

The policy can be modified by changing <*policy*> to one of the following 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.

Capitalization does not matter for the value of <*policy*>.

Beyond the cluster manager version 10 release, the advanced configuration directive `SlurmDrainReasonPolicy` is replaced by a `cmsh` Slurm WLM parameter, `drainreasonpolicy`.

JobsSamplingMetricsInterval directive

Syntax: `AdvancedConfig = {"JobsSamplingMetricsInterval=<number>", ...}`
Default: `60`

`JobsSamplingMetricsInterval` is a parameter of the `AdvancedConfig` (page 528) 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 528) 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.

AddUserScript directive

Syntax: `AdvancedConfig = {"AddUserScript=<path>", ...}`
Default: *none*

`AddUserScript` is a parameter of the `AdvancedConfig` (page 528) directive. If this parameter is set to a path leading to a script, and if a new user is added using `cmsh` or RIMA 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 528) 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 528) directive. If this parameter is set to a path leading to a script, and if an existing user is removed using `cmsh` or RIMA 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 528) 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 528) directive. It sets the timeout value in seconds, for the script set by `RemoveUserScript`.

AdditionalSubmitHosts directive
AdditionalExecHosts directive

Syntax:

`AdvancedConfig = {"AdditionalSubmitHosts=<host1>,<host2>,...", ...}`

or

`AdvancedConfig = {"AdditionalExecHosts=<host1>,<host2>,...", ...}`

Default: *none*

The `AdditionalSubmitHosts` and `AdditionalExecHosts` directives are parameters of the `AdvancedConfig` (page 528) directive.

These directives can be used to make the cluster manager aware of the existence of a submit host or execution host that is outside of the cluster manager control. The directives can be used with the GE workload manager only, so that CMDaemon does not remove such hosts from the workload manager configuration file during a configuration file maintenance run.

AutomaticMountAll directive

Syntax: `AutomaticMountAll=0|1`
Default: `1`

If the `AutomaticMountAll` directive is 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 is to do with attempting to mount devices listed in `/etc/fstab`. It should not be confused with auto-mounting of filesystems, which is 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.

AllowImageUpdateWithAutoMount directive

Syntax: `AdvancedConfig = {"AllowImageUpdateWithAutoMount=0|1|2|3", ...}`
Default: `0`

The `AllowImageUpdateWithAutoMount` directive is a parameter of the `AdvancedConfig` (page 528) 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 7.5.2). It must be set per software image or per node.

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 the cluster manager is discussed in section 3.10. The need for adding the automounted filesystem to `excludelistupdate` is discussed on page 235.

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 528) 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`.

The standard directive `PublicDNS` (page 520) 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 528) 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@rima92 ~]# 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 (1) ciphers`. 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(5) slapd.conf`.

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 528) directive, as well as the `GlobalConfig` (page 529) 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", ...}
or **Syntax:** GlobalConfig = {"JobInformationDisabled=0|1", ...}
Default: JobInformationDisabled=0

The JobInformationDisabled directive is a parameter of the AdvancedConfig (page 528) directive, as well as the GlobalConfig (page 529) directive. If set to 1 it disables job-based monitoring. 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 528) 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 528) 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 528) 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 1 million.

JobInformationChargeBackKeepCount directive

Syntax: `AdvancedConfig = {"JobInformationChargeBackKeepCount=<number>", ...}`
Default: `JobInformationChargeBackKeepCount=1048576`

The `JobInformationChargeBackKeepCount` directive is a parameter of the `AdvancedConfig` (page 528) 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 528) 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 528) 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 528) 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 528) 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 528) 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 528) 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 528) 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 RIMA 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 RIMA 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.

MaxMeasurablesPerProducer directive

Syntax: `AdvancedConfig = { "MaxMeasurablesPerProducer = <`*number*`>", ...}`
Default: `500`

The `MaxMeasurablesPerProducer` directive is a parameter of the `AdvancedConfig` (page 528) directive.

By default there is a software limit of 500 measurables per data producer. If this limit is exceded, 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.

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 the cluster manager 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 3.9.1.

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 File

In the cluster manager, regular node partitioning setups have their global structure defined using an XML schema, which is installed on the head node in `/cm/local/apps/cmd/etc/htdocs/xsd/disks.xsd`.

This schema does not include a hardware RAID description. 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 a schema using the `xmllint` tool:

Example

```
[root@rima92 ~]# xmllint --noout --schema  /cm/local/apps/cmd/etc/htdocs/xsd/disks.xsd \
 /cm/local/apps/cmd/etc/htdocs/disk-setup/slave-diskless.xml
/cm/local/apps/cmd/etc/htdocs/disk-setup/slave-diskless.xml validates
[root@rima92 ~]#
```

XML schema for partitioning

```
<?xml version='1.0'?>

<!--
#
# SPDX-FileCopyrightText: Copyright (c) 2022 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='cephosdassociation' type='xs:string' minOccurs='0' maxOccurs='1'/>
    <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='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 File

If a hardware RAID has already been created outside of the cluster manager, 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 3.9.

If, instead, hardware RAID is to be created and managed by the cluster manager, then it can be specified using an XML schema, stored on the head node in `/cm/local/apps/cmd/etc/htdocs/xsd/raid.xsd`. The full schema specification is listed next, while schema examples are listed in section D.4.1.

Configuration using the cluster manager 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) 2022 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 specification 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>
    <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 example assumes a single disk. Another disk can be added by adding another pair of `<device><device>` tags and filling in the partitioning specifications for the next 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. 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.

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`.

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@rima92 ~]# cmsh
[rima92]% category use default
[rima92->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>512M</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:no\
NamespaceSchemaLocation="schema.xsd">

<device>
  <blockdev>/dev/sda</blockdev>
    <partition id="a1">
      <size>512M</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>512M</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 3.9.3.

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 3.10.3).

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 3.10.3.

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>512M</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>512M</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>512M</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>512M</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>512M</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>512M</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>512M</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>512M</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>512M</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>512M</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 cluster manager 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/slave-one-big-encrypted-partition-ext4.xml`

with minimal `<encryption />` tags added for the root and swap partitions that are to run encrypted:

```
[rima92->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):

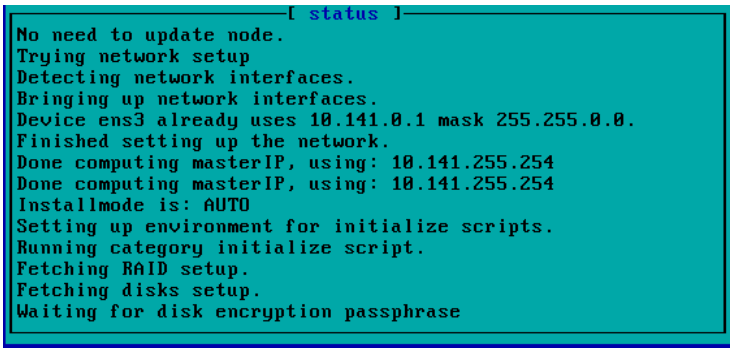


Figure D.1: Node Console Waiting For Disk Encryption Passphrase During Node-installer Run

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

```
[rima92->device[node001]]% installerinteractions
No remote interactions pending.
[rima92->device[node001]]%
Wed Jun  9 12:32:00 2021 [notice] rima92: node001 [ INSTALLING ] (node installer started)
[rima92->device[node001]]%
Wed Jun  9 12:32:11 2021 [notice] rima92: node001 [ INSTALLING ] (waiting for disk encryption passphrase)
[rima92->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

```
[rima92->device[node001]]% installerinteractions -w --confirm
[rima92->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...]

[rima92->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

```
[rima92->device[node001]]% installerinteractions -w --confirm --for 5m
[rima92->device[node001]]% installerinteractions
Hostname  Category  Type                        Timestamp                       Action         Status
--------- --------- --------------------------- ------------------------------- -------------- ----------
node001   default   disk encryption passphrase  Thu Jun 10 16:53:46 2021        Future action  confirmed
[rima92->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

```
[rima92->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] rima92: node001 [ INSTALLING ] (waiting for disk encryption passphrase)
[rima92->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...]
[rima92->device[node001]]% installerinteractions
No remote interactions pending.
[rima92->device[node001]]%
Fri Jun 11 23:09:48 2021 [notice] rima92: node001 [ INSTALLER_CALLINGINIT ] (switching to local root)
[rima92->device[node001]]%
Fri Jun 11 23:10:31 2021 [notice] rima92: node001 [   UP   ]
[rima92->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:

```
[rima92->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 iniitialize and finalize scripts are provided with the default category:

```
[rima92->category[default]]% show | grep ize
Initialize script               <1.46KiB>
Finalize script                 <3.4KiB>
```

E.1 When Are They Used?

The iniitialize and finalize scripts are sometimes used as an alternative configuration option out of a choice of other possible options (section 3.15.1). 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 RIMA View And cmsh

The initialize and finalize scripts are accessible for viewing and editing:

- In RIMA View, via the Node Categories or Nodes window, under the Settings window. The clickpaths 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@rima92 ~]# cmsh
[rima92]% category use default
[rima92->category[default]]% show | grep script
Parameter                      Value
------------------------------ ------------------------------------------------
Finalize script                <1367 bytes>
Initialize script              <0 bytes>
[rima92->category[default]]% set initializescript
```

Example

```
[rima92]% device use node001
[rima92->device[node001]]%
[rima92->device[node001]]% set finalizescript
```

E.3 Environment Variables Available To initialize And finalize Scripts

For the initialize and finalize scripts, node-specific customizations can be made from a script using environment variables.

The environment variables can be listed using the following script as a finalizescript:

Example

```
[rima92->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	RIMA 9.2 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
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

...continues

Table E: Environment Variables For The initialize And Finalize Scripts...continued

Variable	Example Value
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

* 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 RIMA View or cmsh (section 3.10).

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 3.15.4, 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 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 405.

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 3.15.4 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

Metrics, Health Checks, Enummetrics, And Actions

This appendix describes the metrics (section F.1), health checks (section F.2), enummetrics (section 8.2.2), and actions (section F.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.

F.1 Metrics And Their Parameters

A list of metric names can be viewed, for example, for the head node, using `cmsh` as follows (section 8.5.3):

```
[rima92 ~]# cmsh -c "monitoring measurable; list metric"
```

The metrics are listed in this section are classed into 10 kinds:

1. regular metrics (section F.1.1)
2. NFS metrics (section F.1.2)
3. InfiniBand metrics (section F.1.3)
4. monitoring system metrics (section F.1.4)
5. GPU metrics (section F.1.5)
6. Job metrics (section F.1.6)
7. Job queue metrics (section F.1.6)
8. IPMI metrics (section F.1.7)
9. SMART metrics (section F.1.8)
10. Prometheus metrics (section F.1.9)

F.1.1 Regular Metrics

Table F.1.1: List Of Metrics

Metric	Description
AlertLevel	Indicates the healthiness of a device based on severity of events (section 8.2.8). 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
bright_licensed_nodes	Nodes licensed by the cluster manager
bright_pay_by_use_nodes	Nodes that are pay-by-use
bright_used_nodes	Nodes used by the cluster manager
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
CoresTotal	Total number of known cores for all nodes
CoresUp	Number of cores for all nodes marked as UP
CtxtSwitches*	Context switches/s
DevicesClosed	Number of devices not marked as UP or DOWN
DevicesDown	Number of devices marked as DOWN
DevicesTotal	Total number of devices
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.

...continues

Table F.1.1: List Of Metrics...continued

Metric	Description
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
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
fan_speed	Fan speed (RPM), from Redfish
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
HardwareCorruptedMemory	Hardware corrupted memory detected by ECC
IOInProgress†	I/O operations in progress
IOTime*,†	I/O operations time in milliseconds/s
Interrupts *	System interrupts/s
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
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

...continues

Table F.1.1: List Of Metrics...continued

Metric	Description
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
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
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
MemoryAvailable	Available system memory
MemoryFree	Free system memory
MemoryTotal	Total system memory
MemoryUsed	Used system memory
MemoryUsed¶	Used system memory for process, but default for cmd
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.

...continues

Table F.1.1: List Of Metrics...continued

Metric	Description
	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.
OpaHfiTxByteCount*,‡	Intel Omni-Path HFI Transmitted data/s
OpaHfiTxPktsCount*,‡	Intel Omni-Path HFI Transmitted packets/s
OpaHfiRxByteCount*,‡	Intel Omni-Path HFI Received data/s
OpaHfiRxPktsCount*,‡	Intel Omni-Path HFI Received packets/s
PacketsRecv*,‡	Packets/s received
PacketsSent*,‡	Packets/s sent
PageFaults*	Page faults/s
PageIn *	Number of bytes the system has paged in from disk/s
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).
psu_power	Average power supply unit output (W), from Redfish
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
SectorsRead*,†	Sectors/s read successfully/s
SectorsWritten*,†	Sectors/s written successfully
sensor_reading	Temperature sensor reading (C), from Redfish
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, by default CMDaemon
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, by default CMDaemon

...continues

Table F.1.1: List Of Metrics...continued

Metric	Description
TotalCPUIdle	Cluster-wide core usage in idle tasks (sum of all CPUIdle metric percentages)
TotalCPUSystem	Cluster-wide core usage in system mode (sum of all CPUSystem metric percentages)
TotalCPUUser	Cluster-wide core usage in user mode (sum of all CPUUser metric percentages)
TotalGPUPowerUsage	Total GPU power usage, usinggpu_power_usage:total (table F.1.5) over all nodes
TotalGPUUtilization	Average of GPU utilizaion using gpu_utilization:average (table F.1.5) over all nodes
TotalMemory	Sum of MemoryTotal over all nodes
TotalMemoryFree	Cluster-wide total of memory free
TotalMemoryUsed	Cluster-wide total of memory used
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
UniqueUserLogin	Number of unique users logged in
UnmanagedNodesClosed	Number of unmanaged nodes not marked as UP or DOWN
UnmanagedNodesDown	Number of unmanaged nodes marked as DOWN
UnmanagedNodesTotal	Total number of unmanaged nodes
UnmanagedNodesUp	Number of unmanaged nodes marked as UP
Uptime*	System uptime per second. Ie, ideally=1, but in practice has jitter effects
UsedFiles§	Used file inodes on the specified mount point
UsedSpace§	Used space on the specified mount point
UserTime¶	User time used by process, by default CMDaemon
VirtualMemoryUsed¶	Virtual memory used by process, by default CMDaemon
WlmSlotsFree	The number of WLM slots in 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

Table F.1.1: List Of Metrics...continued

Metric	Description
WriteTime*,†	Write time in milliseconds/s (this is a per mille)
Writes*,†	Writes/s completed successfully

* 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, by default cmd, as parameter

F.1.2 NFS Metrics

The NFS metrics are all cumulative. They correspond to nfsstat output, and are shown in table F.1.2.

Table F.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
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

...continues

Table F.1.2: NFS Metrics

NFS Metric	Description
nfs_server_rpc_calls	NFS Server RPC statistics: all calls to NFS and NLM
nfs_server_rpc_xdrcall	NFS Server RPC statistics: malformed XDR calls
nfs_v3_client_access	NFSv3 client statistics: access
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_total	NFSv3 client statistics: total
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
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

F.1.3 InfiniBand Metrics

The available InfiniBand metrics are displayed in table F.1.3.

Table F.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 F.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).

F.1.4 Monitoring System Metrics

The monitoring system metrics correspond to metrics for the monitoring data storage itself. These are shown in table F.1.4.

Table F.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 F.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 the cluster manager format sampling.
Prometheus::DataTranslator::miss	DataTranslator misses

F.1.5 GPU Metrics

The data producer (section 8.2.10) for the GPU metrics of this section is GPUSampler.

There were GPU metrics described earlier on in table F.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.

The device parameter for the GPU metrics in this section 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 F.1.5.

If the cluster has been configured with AMD GPUs (section 7.5 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.

The available GPU health checks for V100 and A100 GPUs are listed in table F.2.1.

Table F.1.5: GPU Metrics

GPU Metric	Description
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_enforced_power_limit	GPU enforced power limit
gpu_mem_clock	GPU memory clock (also for AMD GPU)
gpu_mem_total	GPU framebuffer size (also for AMD GPU)
gpu_mem_used	Amount of GPU memory used (also for AMD GPU)

...continues

Table F.1.5: GPU Metrics...continued

GPU Metric	Description
gpu_mem_utilization	GPU memory used percentage
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_profiling_fp16_active	ratio of cycles the FP16 pipe is active
gpu_profiling_fp64_active	ratio of cycles the FP64 pipe is active
gpu_profiling_pipe_tensor_active	ratio of cycles the tensor (HMMA) pipe is active (A100 only)
gpu_shutdown_temp	GPU shutdown temperature
gpu_slowdown_temp	GPU slowdown temperature
gpu_sm_clock	GPU shader multiprocessor clock (also for AMD GPU)
gpu_temperature	GPU temperature
gpu_temperature_edge	GPU edge temperature (AMD GPU only)
gpu_temperature_junction	GPU junction temperature (AMD GPU only)
gpu_temperature_memory	GPU memory temperature (AMD GPU only)
gpu_utilization	Average GPU utilization percentage. This takes one of the following parameters: • gpu_utilization:gpu<*number*>: the GPU utilization percentage for a GPU device. This takes as its parameter the GPU device (gpu0, gpu1...) as is usual for the GPU metrics in this table • gpu_utilization:average: the GPU utilization percentage as an average over the node. When the average parameter replaces the usual device parameter, it samples the average percentage over all the GPU devices that are in use on that node • gpu_utilization:compute_instance<*number*>: the GPU utilization percentage for a MIG GPU instance 0 (A100 only). The compute instance <*number*> can be from 0 up to 6 for the A100 MIGs.

F.1.6 Job Metrics

Basic Job Metrics

The following table lists some of the most useful job metrics that the cluster manager 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 F.1.6.1: 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`
`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`

...continues

...continued

Job Metric	Description	Cgroup Source File
`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:

```
[rima92->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
[rima92->monitoring->setup[JobSampler]->jobmetricsettings]% set enableadvancedmetrics yes
[rima92->monitoring->setup*[JobSampler*]->jobmetricsettings*]% commit
```

The advanced job metrics are:

Table F.1.6.2: 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`

...continues

...continued

Advanced Job Metric	Description	Cgroup Source File
`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`
`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`

...continues

...continued

Advanced Job Metric	Description	Cgroup Source File
`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 F.1.1: 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

...continues

Table F.1.1: List Of Job Queue Metrics...continued

Metric	Description
CoresUPInQueue	Active cores in queue
EstimatedDelay	Estimated delay time to execute jobs
FailedJobs	Failed completed jobs
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

F.1.7 IPMI Metrics

The IPMI metrics correspond to metrics provided by the BMC devices. The metrics available depend on the manufacturer, and are detected by the cluster manager. The metrics listed in the following table are a limited list of what may be detected on a system.

The data producer (section 8.2.10) for the IPMI metrics is the `ipmi` data producer.

Table F.1.7: IPMI Metrics

IPMI Metric	Description
Current_<*number*>**	Current seen by BMC sensor <*number*>, in amps (file: `sample_ipmi`)
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`)

...continues

Table F.1.7: IPMI Metrics...continued

IPMI Metric	Description
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/

F.1.8 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 the cluster manager. 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 8.2.10) for SMART metrics is the smart data producer.

Table F.1.8: 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

F.1.9 Prometheus Metrics

The data producers for Prometheus metrics are JobSampler, JobMetadataSampler, and some others.

Table F.1.9: 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
`job_blkio_sectors`	Reports the number of sectors transferred to or from specific devices by a cgroup

...continues

Table F.1.9: Prometheus Metrics...continued

Prometheus Metric	Description (for a job, unless asterisked)
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
job_memory_rss_bytes	Anonymous and swap cache, not including tmpfs (shmem), in bytes

...continues

Table F.1.9: Prometheus Metrics...continued

Prometheus Metric	Description (for a job, unless asterisked)
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

F.1.10 Parameters For Metrics

Metrics have the parameters indicated by the left column in the following example:

Example

```
[rima92->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 8.4.3 and 8.5.2

`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 sample age is not considered.

`Maximum`: the value that the y-axis maximum takes in graphs plotted in RIMA 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.[1]

`Minimum`: the value that the y-axis minimum takes in graphs plotted in RIMA 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.[1]

`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`

[1] To clarify the concept, 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.

`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 %

F.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 8.5.3):

```
[rima92 ~]# cmsh -c "monitoring measurable; list healthcheck"
```

Regular health checks (section 8.2.4) are listed and described in section F.2.1.
Redfish health checks are listed and described in section F.2.2.

F.2.1 Regular Health Checks

Table F.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 `latesthealtdata -v` command (section 8.6.3) should show which one(s). After correcting the problem with the service, a reset of the service is normally carried out (section 3.11, page 128).
`Mon::Storage`	Is space available for the monitoring system metrics (section F.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.
`beegfs`	Is BeeGFS properly connected as expected to all the nodes? Unreachable nodes and failed routes are listed in InfoMessages.
`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 591) instead.*

...continued

Table F.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 F.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? 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.
failover	Is the failover status OK?
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 health OK?

...continued

Table F.2.1: List Of Health Checks...continued

Name	**Query** (response is `PASS/FAIL`)
`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?
`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 all up and OK?
`ipmihealth`	Is the BMC (IPMI or iLO) health OK? Uses the script `sample_ipmi`.
`kubernetes childnode`	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 Bright Cluster Manager versions up to 9.2, the healthcheck first checks for MegaCLI, and then StorCLI, and uses the first binary that is detected. For versions later than 9.2, the healthcheck first checks for StorCLI, and then for MegaCLI, and uses the first binary that is detected.*

...continued

Table F.2.1: List Of Health Checks...continued

Name	Query (response is PASS/FAIL)
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.*
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 621) 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 F.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.
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/
‖ Takes GPU device name (gpu0, gpu1, and so on) as parameter. For example: gpu_health_driver:gpu0

F.2.2 Redfish Healthchecks

The available Redfish health checks are displayed in table F.2.2.

Table F.2.2: Redfish Healthchecks

Redfish Healthcheck	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
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)

F.2.3 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 F.1.10.

F.3 Enummetrics

Table F.3: List Of Enummetrics

Name	Query
`DeviceStatus`	What is the status of the device? Possible values are: • `up` • `down` • `closed` • `installing` • `installer_failed` • `installer_rebooting,` • `installer_callinginit` • `installer_unreachable` • `installer_burning` • `burning` • `unknown` • `opening` • `going_down` • `pending` • `no data`

F.4 Actions And Their Parameters

F.4.1 Actions

Table F.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 to the CMD_INFO_MESSAGE environmental variable. Syntax: killprocess <*PID1*[,<*PID2*>,...]> This action is designed to work with rogueprocess (page 618)
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/

F.4.2 Parameters For A Monitoring Action

The default monitoring actions are listed in section 8.4.4.

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 RIMA 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

G

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 G.1), and
- setting the new password in `cmd.conf` (section G.2).

It is also a good idea to do some checking afterwards (section G.3).

G.1 Setting A New Password For The LDAP Server

An encrypted password string can be generated as follows:

```
[root@rima92 ~]# module load openldap
[root@rima92 ~]# 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@rima92 ~]# 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@rima92 ~]# service slapd restart
```

G.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 G.1) is set in `cmd.conf`. It is kept as clear text for the entry for the LDAPPass directive (Appendix C):

```
[root@rima92 ~]# grep LDAPPass /cm/local/apps/cmd/etc/cmd.conf
LDAPPass = "Mysecret1dappassw0rd"
```

CMDaemon is then restarted:

```
[root@rima92 ~]# service cmd restart
```

G.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@rima92 ~]# ldapsearch -x
# extended LDIF
#
# LDAPv3
# base <dc=cm,dc=cluster> (default) with scope subtree
...
```

- root cn without a password (this should fail):

```
[root@rima92 ~]# 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@rima92 ~]#
```

- root cn with a password (this should work):

```
[root@rima92 ~]# ldapsearch -x -D 'cn=root,dc=cm,dc=cluster' -w Mysecret1dappassw0rd
# extended LDIF
#
# LDAPv3
# base <dc=cm,dc=cluster> (default) with scope subtree
...
```

H

Tokens

This appendix describes authorization tokens available for profiles. Profiles are introduced in Section 6.4:

Table H: List Of Tokens

Service and token name	User can...
Service: CMAuth	
GET_PROFILE_TOKEN	Retrieve list of profiles and profile properties
ADD_PROFILE_TOKEN	Add a new profile
UPDATE_PROFILE_TOKEN	Update profile
GET_CMSERVICES_TOKEN	Get a list of available CMDaemon services
Service: CMBeeGFS	
GET_BEEGFS_TOKEN	Get BeeGFS properties
ADD_BEEGFS_TOKEN	Add new BeeGFS property
UPDATE_BEEGFS_TOKEN	Update BeeGFS properties
GET_BEEGFS_NODES_TOKEN	Get BeeGFS nodes properties
Service: CMCeph	
GET_CEPH_TOKEN	Get Ceph properties
ADD_CEPH_TOKEN	Add new Ceph configuration
UPDATE_CEPH_TOKEN	Update Ceph configuration
CEPH_KEYS_TOKEN	Manipulate Ceph keys
CEPH_CLUSTER_SERVICES_TOKEN	List Ceph services
CEPH_SET_DECOMMISSION_TOKEN	Decommission Ceph nodes
CEPH_GET_DECOMMISSION_TOKEN	Get decommissioning information
UPDATE_CEPH_STATE_TOKEN	Update Ceph state
UPDATE_CEPHOSDMAP_TOKEN	Update Ceph OSD remote
GENERIC_CEPH_COMMAND_TOKEN	Run generic Ceph command

...continues

Table H: List Of Tokens...continued

Service and token name	User can...
DRAIN_CEPH_OVERVIEW_TOKEN	See Drain Ceph overview
DRAIN_CEPH_TOKEN	Drain Ceph
Service: CMCert	
ISSUE_CERTIFICATE_TOKEN	Accept certificate request and issue signed certificate
GET_CERTIFICATE_REQUEST_TOKEN	List pending certificate requests
RECREATE_COMPONENT_CERTIFICATE TOKEN	Recreate component certificate
INVALIDATE_COMPONENT_CA_TOKEN	Invalidate component CA
REMOVE_CERTIFICATE_REQUEST_ TOKEN	Cancel certificate request
GET_CERTIFICATE_TOKEN	Get certificate
REVOKE_CERTIFICATE_TOKEN	Revoke a certificate
UNREVOKE_CERTIFICATE_TOKEN	Unrevoke a revoked certificate
REMOVE_CERTIFICATE_TOKEN	Remove a certificate
Service: CMCloud	
GET_CLOUD_PROVIDER_TOKEN	Get cloud provider information
ADD_CLOUD_PROVIDER_TOKEN	Add a new cloud provider
UPDATE_CLOUD_PROVIDER_TOKEN	Update cloud provider settings
EC2_ACCESS_STRING_TOKEN	Get/set Amazon EC2 access string
AZURE_ACCESS_STRING_TOKEN	Get/set Azure access string
OSCLOUD_ACCESS_STRING_TOKEN	Get/set OpenStack access string
GET_CLOUD_REGION_TOKEN	Access Amazon EC2 region
GET_CLOUD_AMI_TOKEN	Access Amazon EC2 AMI
GET_CLOUD_TYPE_TOKEN	Access Amazon instance type
GET_KERNEL_INITRD_MD5SUM_ TOKEN	Retrieve MD5 sum of initial ramdisk
PUT_USERDATA_TOKEN	Set AWS user data in AWS
TERMINATE_NODE_TOKEN	Terminate cloud nodes
GET_AWS_KEY_TOKEN	Retrieve AWS key
CLOUD_DIRECTOR_NEW_IP_TOKEN	Set the new External IP of the cloud director
GET_CONSOLE_OUTPUT_TOKEN	Retrieve the console output of the cloud director for debugging purposes
SET_CLOUDERRORS_TOKEN	Set cloud errors
GET_CLOUD_STATIC_IPS_TOKEN	Get the static IP list of the cloud nodes
SEND_CLOUD_STORAGE_ACTION_ TOKEN	Send cloud storage action
ADD_CLOUD_JOB_DESCRIPTION_TOKEN	Add cloud job description
CANCEL_ANY_CLOUD_JOB_TOKEN	Cancel any cloud job

...continues

Table H: List Of Tokens...continued

Service and token name	User can...
GET_ALL_CLOUD_JOB_DESCRIPTION_TOKEN	Get all cloud job description
GET_CLOUD_JOB_DESCRIPTION_TOKEN	Get cloud job description
SUBMIT_CLOUD_JOB_DESCRIPTION_TOKEN	Submit cloud job description
UPDATE_CLOUD_JOB_DESCRIPTION_TOKEN	Update cloud job description
VPC_MAGIC_CALL_TOKEN	(Internal use only)
ALLOCATE_STATIC_IP_TOKEN	Allocate static IP to cloud director
UNALLOCATE_STATIC_IP_TOKEN	De-allocate static IP
ADD_CMJOB_CONFIG_TOKEN	Add cmjob configuration
GET_CMJOB_CONFIG_TOKEN	Get cmjob configuration
UPDATE_CMJOB_CONFIG_TOKEN	Update cmjob configuration
LIST_ALL_ANF_VOLUMES_TOKEN	List all ANF volumes
SHARE_ANF_VOLUME_TOKEN	Share own ANF volumes
DELETE_ANY_ANF_VOLUME_TOKEN	Delete any ANF volume
USER_MANAGED_ANF_TOKEN	Manage ANF volume
LIST_AlL_FSX_INSTANCES_TOKEN	List all FSX instances
SHARE_FSX_INSTANCE_TOKEN	Share own FSX instances
DELETE_ANY_FSX_INSTANCE_TOKEN	Delete any FSX instance
USER_MANAGED_FSX_TOKEN	Manage FSX volume
Service: CMDevice	
SNMP_STRING_TOKEN	Get ethernet switch SNMP public string
CHASSIS_USER_PASSWORD_TOKEN	Get/set chassis username and password
BMC_USERNAME_PASSWORD_TOKEN	View/set BMC (e.g. HP ilo4, IPMI) username and password
GET_DEVICE_TOKEN	View all device properties
GET_DEVICE_BY_PORT_TOKEN	View list of devices according to the ethernet switch port that they are connected to
ADD_DEVICE_TOKEN	Add a new device
UPDATE_DEVICE_TOKEN	Update device properties
ADD_CATEGORY_TOKEN	Create new category
GET_CATEGORY_TOKEN	Get list of categories
UPDATE_CATEGORY_TOKEN	Update a category property
ADD_CONFIGURATIONOVERLAY_TOKEN	Create new configuration overlay
GET_CONFIGURATIONOVERLAY_TOKEN	Get list of configuration overlays
UPDATE_CONFIGURATIONOVERLAY_TOKEN	Update a configuration overlay property
ADD_NODEGROUP_TOKEN	Add a new nodegroup
GET_NODEGROUP_TOKEN	Get list of nodegroups
UPDATE_NODEGROUP_TOKEN	Update nodegroup properties (e.g. add a new member node)

...continues

Table H: List Of Tokens...continued

Service and token name	User can...
GET_DEVICE_STATUS_TOKEN	Get device status (e.g. UP as well as status string e.g. restart required)
SET_DEVICE_STATUS_TOKEN	Set device status (only via RPC API calls)
POWER_ON_TOKEN	Power on a device using BMC or PDU power control
POWER_OFF_TOKEN	Power off a device
POWER_CANCEL_TOKEN	Power cancel operation
POWER_CYCLE_TOKEN	Power reset a device
POWER_STATUS_TOKEN	Get power status e.g on or off
POWER_RESULT_TOKEN	Get the result of the previous power command e.g. failed
REPORT_POWER_STATUS_TOKEN	Report power operation history
SHUTDOWN_NODE_TOKEN	Shutdown a remote node managed by CMDaemon
REBOOT_NODE_TOKEN	Reboot a remote a node
FORCE_RECONNECT_TOKEN	Force remote client to reconnect (RPC API)
NODE_IDENTIFY_TOKEN	Identify a node (RPC API, used by node installer)
NODE_GET_MOUNTPOINTS_TOKEN	Get list of mountpoints defined for a node
PPING_TOKEN	Run parallel ping
START_BURN_TOKEN	Start burn
STOP_BURN_TOKEN	Stop burn
BURN_STATUS_TOKEN	Get burn status
REQUEST_BURN_TOKEN	Request burn token
CANCEL_BURN_TOKEN	Cancel burn token
GET_BURN_LOG_TOKEN	Retrieve burn log
GET_SYNC_LOG_TOKEN	Get rsync provisioning log
GET_SYNC_INFO_TOKEN	Get rsync information
GET_SYNC_SOURCE_TOKEN	Get rsync source
GET_SYNC_TARGET_TOKEN	Get rsync target
GET_PORT_BY_MAC_TOKEN	Determine to which switch port a given MAC is connected to.
GET_SYSINFO_COLLECTOR_TOKEN	Get information about a node (executes dmidecode)
PUT_SYSINFO_COLLECTOR_TOKEN	Put information about a node
REMOVE_SYSINFO_COLLECTOR_TOKEN	Remove information about a node
UPDATE_SYSINFO_COLLECTOR_TOKEN	Update information about a node
GET_EXCLUDE_LIST_TOKEN	Retrieve the various exclude lists
ADD_REMOTE_NODE_INSTALLER_ INTERACTION_TOKEN	Add a node-installer interaction (Used by CMDaemon)
REMOVE_REMOTE_NODE_ INSTALLER_INTERACTION_TOKEN	Remove a node installer interaction
GET_REMOTE_NODE_INSTALLER_ INTERACTIONS_TOKEN	Get list of pending installer interactions
UPDATE_REMOTE_NODE_	Update installer

...continues

Table H: List Of Tokens...continued

Service and token name	User can...
INSTALLER_INTERACTIONS_TOKEN	interactions (e.g. confirm full provisioning)
ADD_FILE_WRITE_INFO_TOKEN	Add filewriteinfo
GET_FILE_WRITE_INFO_TOKEN	Get filewriteinfo
ADD_LITENODE_TOKEN	Add lite node
MONITORING_LITE_TOKEN	Monitoring lite node
ADD_NODE_HIERARCHY_RULE_TOKEN	Add node hierarchy rule
GET_NODE_HIERARCHY_RULE_TOKEN	Get node hierarchy rule
UPDATE_NODE_HIERARCHY_RULE_TOKEN	Update node hierarchy rule
ADD_REPORT_QUERY_TOKEN	Add report query
GET_REPORT_QUERY_TOKEN	Get report query
REPORT_QUERY_TOKEN	Show report query
UPDATE_REPORT_QUERY_TOKEN	Update report query
ADD_UNMANAGED_NODE_CONFIGURATION_TOKEN	Add unmanaged node configuration
GET_UNMANAGED_NODE_CONFIGURATION_TOKEN	Get unmanaged node configuration
UPDATE_UNMANAGED_NODE_CONFIGURATION_TOKEN	Update unmanaged node configuration
BACKUP_INFORMATION_TOKEN	Show backup information
GET_BACKUP_INFO_TOKEN	Get backup information
REMOVE_BACKUP_TOKEN	Remove backup information
SET_BACKUP_INFO_TOKEN	Set backup information
BIOS_APPLY_TOKEN	BIOS apply token
BIOS_FETCH_TOKEN	BIOS fetch token
CHECK_FABRIC_PEER_INTERFACES_TOKEN	Check fabric peer interfaces
FABRIC_APPLY_TOKEN	Apply fabric
FABRIC_FETCH_TOKEN	Fetch fabric
FABRIC_NODE_STATUS_TOKEN	Show fabric node status
FABRIC_PASSWORD_TOKEN	Manage fabric password
CHECK_REMOTE_MOUNT_TOKEN	Check remount mount
CLEAR_DISK_ENCRYPTION_PASSPHRASE_TOKEN	Clear disk encryption passphrase
FIRMWARE_FLASH_TOKEN	Flash firmware
FIRMWARE_INFO_TOKEN	Get firmware information
FIRMWARE_UPLOAD_TOKEN	Upload firmware
GET_DISKSETUP_TOKEN	Get disksetup
GET_GPU_MIG_TOKEN	Get GPU MIG
SET_GPU_MIG_TOKEN	Set GPU MIG
GET_NODE_ACCELERATOR_TOKEN	Get node accelerator count
SET_NODE_ACCELERATOR_TOKEN	Set node accelerator
GET_SCRIPT_ENVIRONMENT_TOKEN	Get script environment
GET_TFTPBOOT_FILE_INFORMATION_TOKEN	Get tftpboot firmware information
GET_WIREGUARD_INFO_TOKEN	Get wireguard info

...continues

Table H: List Of Tokens...continued

Service and token name	User can...
Service: CMEtcd	
ADD_ETCD_TOKEN	Add etcd
GET_ETCD_TOKEN	Get etcd
UPDATE_ETCD_TOKEN	Update etcd
Service: CMKeyvalue	
SUBSCRIBE_TO_KEY_VALUE_CHANGED_EVENTS_TOKEN	Subscribe to key value changed events
KEY_VALUE_STORE_GET_KEYS_TOKEN	Get key value store get keys
KEY_VALUE_STORE_GET_PAIRS_TOKEN	Get key value store get pairs
KEY_VALUE_STORE_COMMIT_OPERATIONS_TOKEN	Get key value store commit operations
KEY_VALUE_STORE_PURGE_TOKEN	Get key value store purge
Service: CMKube	
ADD_KUBE_TOKEN	Add Kube
GET_KUBE_TOKEN	Get Kube
UPDATE_KUBE_TOKEN	Update Kube
DRAIN_KUBE_TOKEN	Drain Kube
DRAIN_KUBE_OVERVIEW_TOKEN	Drain Kube overview
KUBE_MANAGED_LABELS_TOKEN	Get Kube managed labels response
UPDATE_KUBE_INITIALIZATION_STATE_TOKEN	Update Kube initialization state
Service: CMGui	
GET_CLUSTER_OVERVIEW_TOKEN	Get cluster overview
GET_KUBE_OVERVIEW_TOKEN	Get kube cluster overview
GET_NODE_OVERVIEW_TOKEN	Get node overview
GET_NETSWITCH_OVERVIEW_TOKEN	Get switch overview
GET_PDU_OVERVIEW_TOKEN	Get PDU overview
GET_NODE_STATUS_TOKEN	Get node status
GET_CEPH_OVERVIEW_TOKEN	Get Ceph overview
Service: CMJob	
GET_OWN_JOB_TOKEN	Get list of own jobs that are currently running
HOLD_OWN_JOB_TOKEN	Place own job on hold
REQUEUE_OWN_JOB_TOKEN	Requeue own job
SUSPEND_OWN_JOB_TOKEN	Suspend own job
RESUME_OWN_JOB_TOKEN	Resume own job
RELEASE_OWN_JOB_TOKEN	Release own job
UPDATE_OWN_JOB_TOKEN	Update own job

...continues

Table H: List Of Tokens...continued

Service and token name	User can...
GET_JOB_TOKEN	Get list of jobs that are currently running
HOLD_JOB_TOKEN	Place a job on hold
REQUEUE_JOB_TOKEN	Requeue a job
SUSPEND_JOB_TOKEN	Suspend a job
RESUME_JOB_TOKEN	Resume suspended job
RELEASE_JOB_TOKEN	Release a held job
UPDATE_JOB_TOKEN	Update job run-timer parameters
SUBMIT_JOB_TOKEN	Submit a job using JSON
GET_JOBQUEUE_TOKEN	Retrieve list of job queues and properties
UPDATE_JOBQUEUE_TOKEN	Modify job queues
ADD_JOBQUEUE_TOKEN	Add a new job queue
GET_PE_TOKEN	Get list of SGE parallel environments
DRAIN_TOKEN	Drain a node
DRAIN_OVERVIEW_TOKEN	Obtain list of drained nodes
GET_JOBINFO_TOKEN	Get job information
GET_OWN_JOBINFO_TOKEN	Get own job information
SET_PERSISTENT_JOBINFO_TOKEN	Set persistent job information
REMOVE_JOBINFO_TOKEN	Remove job information
JOB_STARTED_ENDED_TOKEN	Show job started/ended
GET_WLM_CLUSTER_TOKEN	Get WLM cluster
ADD_WLM_CLUSTER_TOKEN	Add WLM cluster
UPDATE_WLM_CLUSTER_TOKEN	Update WLM cluster
UPDATE_WLM_CLUSTER_SERVER_KEY_TOKEN	Update WLM cluster server key
UPDATE_WLM_CLUSTER_SERVER_NODE_TOKEN	Update WLM cluster server node
CHARGE_BACK_TOKEN	Show chargeback
CHARGE_BACK_BY_KEY_TOKEN	Show chargeback by key
GET_CHARGE_BACK_REQUEST_TOKEN	Get chargeback request
ADD_CHARGE_BACK_REQUEST_TOKEN	Add chargeback request
UPDATE_CHARGE_BACK_REQUEST_TOKEN	Update chargeback request
GET_WLM_POWER_SAVING_TOKEN	Get WLM power saving status
SAMPLE_NOW_TOKEN	Sample now
Service: CMMain	
GET_LICENSE_INFO_TOKEN	Retrieve information about BCM license
GET_VERSION_TOKEN	Get CMDaemon version and revision
GET_SERVER_STATUS_TOKEN	Head node status (e.g. ACTIVE, BECOMEACTIVE etc.)
GET_CLUSTER_SETUP_TOKEN	Get cluster configuration
PING_TOKEN	TCP SYN ping managed devices
SAVE_FILE_TOKEN	Save a file on a remote node

...continues

Table H: List Of Tokens...continued

Service and token name	User can...
UPDATE_SELF_TOKEN	Update a category property
CMDAEMON_FAILOVER_TOKEN	Set CMDaemon failover condition achieved
CMDAEMON_QUORUM_TOKEN	Set CMDaemon quorum achieved
GENERIC_CALL_TOKEN	Make a generic call
REPORT_CRITICAL_ERROR_TOKEN	View critical error report
SET_SERVICESTATE_TOKEN	Set the state of a service
GET_SERVICESTATE_TOKEN	Get the state of a service
GET_BACKGROUND_TASKS_TOKEN	Get background tasks
UPDATE_BACKGROUND_TASK_TOKEN	Update background task
IMPORT_ENTITY_TOKEN	Import an entity
CM_SETUP_EXECUTE_TOKEN	Execute
CM_SETUP_GET_EXECUTION_TOKEN	Get execution
CM_SETUP_REMOVE_EXECUTION_TOKEN	Remove execution
GET_ALL_ACTIVE_PASSIVE_UP_KEYS_TOKEN	Get keys for all active and passive nodes that are up
GET_FROZEN_FILES_TOKEN	Get frozen files
GET_HARDWARE_OVERVIEW_TOKEN	Get frozen files
GET_MANAGED_FILE_PATHS_TOKEN	Get managed file paths
GET_XSD_SCHEMA_TOKEN	Get XSD schema
STORE_LDAP_CERTIFICATES_TOKEN	Store LDAP certificates
Service: CMMon	
MONITORING_MANAGE_TOKEN	Manage monitoring configuration settings
GET_PICKUP_INTERVAL_TOKEN	Get pickup interval
REQUEST_PICKUP_INTERVAL_TOKEN	Request monitoring pickup interval
PREJOB_TOKEN	Set up prejob check
ADD_ENTITY_MEASURABLE_TOKEN	Add entity measurable
GET_ENTITY_MEASURABLE_TOKEN	Get entity measurable
ADD_LABELED_ENTITY_TOKEN	Add labeled entity
GET_LABELED_ENTITY_TOKEN	Get labeled entity
NEW_LABELED_ENTITY_TOKEN	Use new labeled entity
UPDATE_LABELED_ENTITY_TOKEN	Update labeled entity
ADD_MONITORING_ACTION_TOKEN	Add monitoring action
GET_MONITORING_ACTION_TOKEN	Get monitoring action
UPDATE_MONITORING_ACTION_TOKEN	Update monitoring action
ADD_MONITORING_CONSOLIDATOR_TOKEN	Add monitoring consolidator
GET_MONITORING_CONSOLIDATOR_TOKEN	Get monitoring consolidator
UPDATE_MONITORING_CONSOLIDATOR_TOKEN	Update monitoring consolidator
ADD_MONITORING_DATA_PRODUCER_TOKEN	Add monitoring data producer
GET_MONITORING_DATA_PRODUCER_TOKEN	Get monitoring data producer

...continues

Table H: List Of Tokens...continued

Service and token name	User can...
UPDATE_MONITORING_DATA_PRODUCER_TOKEN	Update monitoring data producer
ADD_MONITORING_MEASURABLE_TOKEN	Add monitoring measurable
CREATE_MONITORING_MEASURABLE_TOKEN	Create monitoring measurable
GET_MONITORING_MEASURABLE_TOKEN	Get monitoring measurable
UPDATE_MONITORING_MEASURABLE_TOKEN	Update monitoring measurable
ADD_MONITORING_STANDALONE_TOKEN	Add monitoring standalone
GET_MONITORING_STANDALONE_TOKEN	Get monitoring standalone
UPDATE_MONITORING_STANDALONE_TOKEN	Update monitoring standalone
ADD_MONITORING_TRIGGER_TOKEN	Add monitoring trigger
GET_MONITORING_TRIGGER_TOKEN	Get monitoring trigger
UPDATE_MONITORING_TRIGGER_TOKEN	Update monitoring trigger
GET_TRIGGER_DATA_TOKEN	Get monitoring trigger evaluation data
ADD_PROMETHEUS_QUERY_TOKEN	Add Prometheus query
GET_PROMETHEUS_QUERY_TOKEN	Get Prometheus query
UPDATE_PROMETHEUS_QUERY_TOKEN	Update Prometheus query
EXECUTE_QUERY_TOKEN	Execute Prometheus query
EXECUTE_QUERY_BY_KEY_TOKEN	Execute Prometheus query by key
PROMETHEUS_EXPORTER_TOKEN	Show Prometheus exporter
PROMETHEUS_METRIC_TOKEN	Show Prometheus metric
MONITORING_PREPARE_CONTINUE_BACKUP_TOKEN	Monitor prepare continue backup
GET_INFO_MESSAGE_TOKEN	Get info message
OFFLOAD_INFORMATION_TOKEN	Get offload information
PUT_OFFLOAD_INFORMATION_TOKEN	Put offload information
PLOT_TOKEN	Request plot
Service: CMNet	
GET_NETWORK_TOKEN	Get network settings
ADD_NETWORK_TOKEN	Add network settings
UPDATE_NETWORK_TOKEN	Update network settings
Service: CMPart	
GET_PARTITION_TOKEN	Get partition settings
ADD_PARTITION_TOKEN	Add partition settings
UPDATE_PARTITION_TOKEN	Update partition settings
GET_RACK_TOKEN	Get rack settings
ADD_RACK_TOKEN	Add rack settings
UPDATE_RACK_TOKEN	Update rack settings
GET_SOFTWAREIMAGE_TOKEN	Get softwareimage settings
ADD_SOFTWAREIMAGE_TOKEN	Add softwareimage settings
UPDATE_SOFTWAREIMAGE_TOKEN	Update softwareimage settings
REMOVE_SOFTWAREIMAGE_TOKEN	Remove softwareimage settings
ADD_SOFTWAREIMAGE_FILE_SELECTION_TOKEN	Add softwareimage file selection
GET_SOFTWAREIMAGE_FILE_SELECTION_TOKEN	Get softwareimage file selection
REMOVE_SOFTWAREIMAGE_FILE_SELECTION_TOKEN	Remove softwareimage file selection
UPDATE_SOFTWAREIMAGE_FILE_SELECTION_TOKEN	Update softwareimage file selection
UPDATEPROVISIONERS_TOKEN	Updateprovisioners settings
UPDATE_PROVISIONING_NODE_TOKEN	Update provisioning node
CMDAEMON_FAILOVER_STATUS_TOKEN	Obtain status of failover

I

Understanding Consolidation

I.1 Introduction

Consolidation is discussed in the sections on using consolidation in the Monitoring chapter (sections 8.4.3 and 8.5.2).

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 344) 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.

I.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.

I.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 RIMA 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.

I.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

```
[rima92->device[rima92]]% 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

```
[rima92->device[rima92]]% 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
...
[rima92->device[rima92]]% 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
[rima92->device[rima92]]% dumpmonitoringdata --consolidationinterval 1w -7d now forks
Timestamp                  Value                Info
-------------------------- -------------------- ----------
2019/01/08 11:15:12.194    2.59113 processes/s
```

J

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 (`rima92`) 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@rima92 ~]# 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 8.6.2, page 373) of `cmsh`.

J.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 F.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@rima92 ~]# cmsh
[rima92]% monitoring setup
[rima92->monitoring->setup]% nodeexecutionfilters dmesg; list; ..;..
Type         Name (key)               Filter                   Filter operation
------------ ------------------------ ------------------------ ----------------
[rima92->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:

```
[rima92->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

```
[rima92->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>
...
```

J.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:

```
[rima92->monitoring->setup]% nodes dmesg
node001..node004,rima92
```

J.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:

```
[rima92->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
rima92     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,rima92` for this reference cluster.

J.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 637 can be used to define several custom metrics according to what nodes run the script and what nodes are targeted by the script.

J.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:

```
[rima92->monitoring->setup]% add metric all_for_self
[rima92->monitoring->setup*[all_for_self*]]% set consolidator none
[rima92->monitoring->setup*[all_for_self*]]% set script /cm/shared/fm.sh
[rima92->monitoring->setup*[all_for_self*]]% set class Test
[rima92->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 RIMA 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:

```
[rima92->monitoring->setup[all_for_self]]% exit
[rima92->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
rima92     all_for_self  Test  63     0.144s  Sampled on rima92 for rima92
```

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
[rima92->monitoring->setup]% add metric some_for_self
[rima92->monitoring->setup*[some_for_self*]]% set consolidator none
[rima92->monitoring->setup*[some_for_self*]]% set class Test
[rima92->monitoring->setup*[some_for_self*]]% set script /cm/shared/fm.sh
[rima92->...*[some_for_self*]]% nodeexecutionfilters
[rima92->...*[some_for_self*]->nodeexecutionfilters]% add node some_nodes
[rima92->...->nodeexecutionfilters*[some_nodes*]]% set nodes node001 node002
[rima92->...->nodeexecutionfilters*[some_nodes*]]% show
Parameter                      Value
------------------------------ ------------------------------------------------
Filter operation               Include
Name                           some_nodes
Nodes                          node001,node002
Revision
Type                           Node
[rima92->...->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:

```
[rima92->...->nodeexecutionfilters[some_nodes]]% ..; ..; ..
[rima92->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
[rima92->monitoring->setup]% add metric from_head_for_some_others
[rima92->monitoring->setup*[from_head_for_some_others*]]% set consolidator none
[rima92->monitoring->setup*[from_head_for_some_others*]]% set script /cm/shared/fm.sh
[rima92->monitoring->setup*[from_head_for_some_others*]]% set class Test
[rima92->...*[from_head_for_some_others*]]% nodeexecutionfilters
[rima92->...*[from_head_for_some_others*]->nodeexecutionfilters]% active
Added active resource filter
[rima92->...*[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
[rima92->monitoring->setup*[from_head_for_some_others*]->nodeexecutionfilters]% ..
[rima92->...*[from_head_for_some_others*]]% executionmultiplexers
```

```
[rima92->...*[from_head_for_some_others*]->executionmultiplexers]% add node other_nodes
[rima92->..._for_some_others*]->executionmultiplexers*[other_nodes*]]% set nodes node003 node004
[rima92->...*[from_head_for_some_others*]->executionmultiplexers*[other_nodes*]]% show
Parameter                      Value
------------------------------ ------------------------------------------------
Filter operation               Include
Name                           other_nodes
Nodes                          node003,node004
Revision
Type                           Node
[rima92->..._for_some_others*]->executionmultiplexers*[other_nodes*]]% commit; ..; ..;..
[rima92->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

```
[rima92->monitoring->setup]% device monitoringresources rima92
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:

```
[rima92->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 rima92 for node003
node004      from_head_for_some_others  Test 44    0.067s Sampled on rima92 for node004
```

The `nodes` command confirms that the head node, `rima92` is the filtered node, that is, the only node(s) running the metric:

```
[rima92->monitoring->setup]% nodes from_head_for_some_others
rima92
```

J.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 `rima92`. 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:

```
[rima92->monitoring->setup]% nodeexecutionfilters from_head_for_some_others
[rima92->...[from_head_for_some_others]->nodeexecutionfilters]% list
Type         Name (key)              Filter                  Filter operation
------------ ----------------------- ----------------------- ----------------
```

```
Resource     Active head node         Active                    Include
[rima92->...[from_head_for_some_others]->nodeexecutionfilters]% remove active head node
[rima92->...*[from_head_for_some_others*]->nodeexecutionfilters*]% add node head
[rima92->...*[from_head_for_some_others*]->nodeexecutionfilters*[head*]]% set nodes rima92
[rima92->...*[from_head_for_some_others*]->nodeexecutionfilters*[head*]]% show
Parameter                      Value
------------------------------ ------------------------------------------------
Filter operation               Include
Name                           head
Nodes                          rima92
Revision
Type                           Node
[rima92->...*[from_head_for_some_others*]->nodeexecutionfilters*[head*]]% commit
[rima92->...[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:

```
[rima92->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 rima92 for node003
node004      from_head_for_some_others  Test  12    0.07s  Sampled on rima92 for node004
[rima92->monitoring->setup]% nodes
rima92
```

J.4 Excessive Sampling

If another node is appended to the node in the filter, then a warning comes up.

```
[rima92->monitoring->setup]% nodeexecutionfilters from_head_for_some_others
[rima92->...[from_head_for_some_others]->nodeexecutionfilters]% append head nodes node003
[rima92->...*[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


[rima92->...[from_head_for_some_others]->nodeexecutionfilters]% show head

Parameter              Value
---------------------- ------------------------------------------------------
Filter operation       Include
Name                   head
Nodes                  node003,rima92
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 the cluster manager just goes ahead with setting up the filter/muliplex system according to what the administrator has specified. The `nodes` and `samplingnow` commands now show:

```
[rima92->...[from_head_for_some_others]->nodeexecutionfilters]% ..;..
[rima92->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 rima92 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 rima92 for node004
[rima92->monitoring->setup]% nodes from_head_for_some_others
node003,rima92
```

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.

J.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:

```
[rima92->monitoring->setup[dmesg]->nodeexecutionfilters]% add<TAB><TAB>
category  lua       node      overlay   resource  type
```

J.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

```
[rima92->monitoring->setup[<data producer>]->nodeexecutionfilters]% add lua lua-filter
[rima92->...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

```
[rima92->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@rima92 ~]# cat node001.lua
self = {}
self.hostname = "node001"
self.category = "default"

[root@rima92 ~]# cat rima92.lua
self = {}
self.hostname = "rima92"
```

This is in addition to the original filter script:

Example

```
[root@rima92 ~]# 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@rima92 ~]# lua
Lua 5.1.4  Copyright (C) 1994-2015 Lua.org, PUC-Rio
> dofile('rima92.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@rima92 ~]# cat ib-slurm-filter.lua
if self == nil then
  return false
end

on_ib_network = false
```

```
for index, interface in ipair(self.interface) 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@rima92 ~]# 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.

K

A Tree View Of cmsh

K.1 Modes

A 3-level tree of the modes in cmsh is:

```
|-- beegfs
|-- category
|   |-- biossettings
|   |-- bmcsettings
|   |-- fsexports
|   |-- fsmounts
|   |-- gpusettings
|   |-- kernelmodules
|   |-- roles
|   |   |-- advancedsettings
|   |   |-- cgroups
|   |   |-- commsettings
|   |   |-- configs
|   |   |-- configurations
|   |   |-- connectionsettings
|   |   |-- engines
|   |   |-- environments
|   |   |-- excludelistsnippets
|   |   |-- genericresources
|   |   |-- interfaces
|   |   |-- logsettings
|   |   |-- momsettings
|   |   |-- nginxreverseproxy
|   |   |-- nodecustomizations
|   |   |-- openports
|   |   |-- osdconfigurations
|   |   |-- policies
|   |   |-- provisioningassociations
|   |   |-- resourceproviders
|   |   |-- spawner
|   |   |-- storagedrivers
|   |   |-- storagebackends
|   |   `-- zones
|   |-- selinuxsettings
|   |-- services
|   `-- staticroutes
|-- ceph
|   `-- pools
```

```
|-- cert
|-- cloud
|   |-- extensions
|   |-- regions
|   |-- types
|   `-- vpcs
|-- cmjob
|   |-- cloudjobs
|   `-- storagenodepolicies
|-- configurationoverlay
|   |-- customizations
|   `-- roles
|       |-- advancedsettings
|       |-- cgroups
|       |-- commsettings
|       |-- configs
|       |-- configurations
|       |-- connectionsettings
|       |-- engines
|       |-- environments
|       |-- excludelistsnippets
|       |-- genericresources
|       |-- interfaces
|       |-- logsettings
|       |-- momsettings
|       |-- nginxreverseproxy
|       |-- nodecustomizations
|       |-- openports
|       |-- osdconfigurations
|       |-- policies
|       |-- provisioningassociations
|       |-- resourceproviders
|       |-- spawner
|       |-- storagedrivers
|       |-- storagebackends
|       `-- zones
|-- device
|   |-- biosettings
|   |-- bmcsettings
|   |-- cloudsettings
|   |-- dellsettings
|   |-- fsexports
|   |-- fsmounts
|   |-- gpusettings
|   |-- interfaces
|   |-- kernelmodules
|   |-- keyvaluesettings
|   |-- roles
|   |   |-- advancedsettings
|   |   |-- cgroups
|   |   |-- commsettings
|   |   |-- configs
|   |   |-- configurations
|   |   |-- connectionsettings
|   |   |-- engines
```

```
|   |   |-- environments
|   |   |-- excludelistsnippets
|   |   |-- genericresources
|   |   |-- interfaces
|   |   |-- logsettings
|   |   |-- momsettings
|   |   |-- nginxreverseproxy
|   |   |-- nodecustomizations
|   |   |-- openports
|   |   |-- osdconfigurations
|   |   |-- policies
|   |   |-- provisioningassociations
|   |   |-- resourceproviders
|   |   |-- spawner
|   |   |-- storagedrivers
|   |   |-- storagebackends
|   |   `-- zones
|   |-- selinuxsettings
|   |-- services
|   |-- snmpsettings
|   `-- staticroutes
|-- edgesite
|-- etcd
|-- fspart
|   `-- excludelistsnippets
|-- group
|-- hierarchy
|   |-- sources
|   `-- targets
|-- keyvaluestore
|-- kubernetes
|   |-- appgroups
|   |   `-- applications
|   |-- labelsets
|   `-- users
|-- main
|-- monitoring
|   |-- action
|   |-- consolidator
|   |   `-- consolidators
|   |-- labeledentity
|   |-- measurable
|   |-- query
|   |   `-- drilldown
|   |-- report
|   |-- setup
|   |   |-- executionmultiplexers
|   |   |-- jobmetricsettings
|   |   |-- nodeexecutionfilters
|   |   `-- rules
|   |-- standalone
|   `-- trigger
|       `-- expression
|-- network
|-- nodegroup
```

```
|-- partition
|   |-- archos
|   |-- bmcsettings
|   |-- burnconfigs
|   |-- fabrics
|   |-- failover
|   |-- failovergroups
|   |-- provisioningsettings
|   |-- resourcepools
|   |-- selinuxsettings
|   `-- snmpsettings
|-- process
|-- profile
|-- rack
|-- session
|-- softwareimage
|   |-- kernelmodules
|   `-- selection
|-- task
|-- unmanagednodeconfiguration
|   |-- bmcsettings
|   `-- keyvaluesettings
|-- user
|   `-- projectmanager
`-- wlm
    |-- cgroups
    |-- chargeback
    |-- jobqueue
    |-- jobs
    |-- ocisettings
    |-- parallelenvironments
    |-- pelogs
    `-- placeholders
```