©2022 Bright Computing, Inc. All Rights Reserved. This manual or parts thereof may not be reproduced in any form unless permitted by contract or by written permission of Bright Computing, Inc.

**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 Novell, Inc. NVIDIA, CUDA, GPUDirect, HPC SDK, NVIDIA DGX, NVIDIA Nsight, NVLink and Tesla are registered trademarks of NVIDIA Corporation. FLEXlm is a registered trademark of Flexera Software, Inc. PBS Professional, PBS Pro, 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. Bright Computing, Inc. shall not be liable for technical or editorial errors or omissions which may occur in this document. Bright Computing, Inc. shall not be liable for any damages resulting from the use of this document.

**Limitation of Liability and Damages Pertaining to Bright Computing, Inc.**

The NVIDIA Bright Cluster Manager product principally consists of free software that is licensed by the Linux authors free of charge. Bright Computing, Inc. shall have no liability nor will Bright Computing, Inc. provide any warranty for the NVIDIA Bright 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 NVIDIA Bright 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 NVIDIA Bright 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 .............................................................................. xvii
0.2 About This Manual ................................................................ xvii
0.3 About The Manuals In General ............................................... xvii
0.4 Getting Administrator-Level Support ....................................... xviii
0.5 Getting Professional Services .................................................. xviii

1 Introduction .................................................................................. 1
  1.1 NVIDIA Bright 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 ........ 3

2 Cluster Management With NVIDIA Bright 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 ............................................................... 7
    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 .................................. 9
    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 ................................................................. 13
    2.3.4 Profiles ...................................................................... 14
  2.4 Bright View GUI .................................................................... 15
    2.4.1 Installing The Cluster Management GUI Service ............ 15
    2.4.2 Navigating The Cluster With Bright View ...................... 16
  2.5 Cluster Management Shell ...................................................... 19
    2.5.1 Invoking cmsh ............................................................. 19
    2.5.2 Levels, Modes, Help, And Commands Syntax In cmsh ........ 23
    2.5.3 Working With Objects .................................................. 27
# 2.5.4 Accessing Cluster Settings ........................................ 38
# 2.5.5 Advanced \texttt{cmsh} Features ............................... 38

## 2.6 Cluster Management Daemon ..................................... 50
### 2.6.1 Controlling The Cluster Management Daemon ............... 51
### 2.6.2 Configuring The Cluster Management Daemon ............... 52
### 2.6.3 CMDaemon Versions ........................................ 52
### 2.6.4 Configuring The Cluster Management Daemon Logging Facilities .......... 53
### 2.6.5 Configuration File Modification, And The \texttt{FrozenFile} Directive .......... 54
### 2.6.6 Configuration File Conflicts Between The Standard Distribution And The Cluster Manager For Generated And Non-Generated Files .......... 55
### 2.6.7 CMDaemon Lite ........................................ 55

# 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 .............................. 71
### 3.2.4 Tools For Viewing Cluster Connections ..................... 83

## 3.3 Configuring Bridge Interfaces .................................. 85

## 3.4 Configuring VLAN interfaces ................................... 86
### 3.4.1 Configuring A VLAN Interface Using \texttt{cmsh} .............. 86
### 3.4.2 Configuring A VLAN Interface Using Bright View ............ 87

## 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 ............... 89
### 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 Further Notes On Bonding ................................ 90

## 3.6 Configuring InfiniBand And Omni-Path Interfaces .............. 91
### 3.6.1 Installing Software Packages ................................ 91
### 3.6.2 Subnet Managers ....................................... 91
### 3.6.3 InfiniBand Network Settings ................................ 92
### 3.6.4 Enabling SNMP ........................................ 93
### 3.6.5 Verifying Connectivity ................................... 93

## 3.7 Configuring BMC (IPMI/iLO/DRAC/CIMC/Redfish) Interfaces ........ 95
### 3.7.1 BMC Network Settings .................................... 95
### 3.7.2 BMC Authentication ....................................... 97
### 3.7.3 Interfaces Settings ...................................... 98
### 3.7.4 Identification With A BMC ................................ 99

## 3.8 Configuring Switches And PDUs .................................. 99
### 3.8.1 Configuring With The Manufacturer’s Configuration Interface .......... 99
# Table of Contents

5.4.2 Deciding Or Selecting Node Configuration ........................................ 213  
5.4.3 Starting Up All Network Interfaces ................................................ 224  
5.4.4 Determining Install-mode Type And Execution Mode .......................... 225  
5.4.5 Running Initialize Scripts .............................................................. 231  
5.4.6 Checking Partitions, RAID Configuration, Mounting Filesystems ........... 231  
5.4.7 Synchronizing The Local Drive With The Software Image ...................... 232  
5.4.8 Writing Network Configuration Files ................................................ 236  
5.4.9 Creating A Local `/etc/fstab` File .................................................. 236  
5.4.10 Installing GRUB Bootloader ........................................................... 236  
5.4.11 Running Finalize Scripts ............................................................... 238  
5.4.12 Unloading Specific Drivers ............................................................ 239  
5.4.13 Switching To The Local `init` Process ............................................ 239  
5.5 Node States ......................................................................................... 239  
5.5.1 Node States Icons In Bright View .................................................... 239  
5.5.2 Node States Shown In `cmsh` ............................................................ 240  
5.5.3 Node States Indicating Regular Start Up .......................................... 240  
5.5.4 Node States That May Indicate Problems .......................................... 241  
5.6 Updating Running Nodes ....................................................................... 243  
5.6.1 Updating Running Nodes: Configuration With `excludelistupdate` ........ 244  
5.6.2 Updating Running Nodes: With `cmsh` Using `imageupdate` ............... 251  
5.6.3 Updating Running Nodes: With Bright View Using the `update node` Option 251  
5.6.4 Updating Running Nodes: Considerations .......................................... 251  
5.7 Adding New Nodes ................................................................................ 252  
5.7.1 Adding New Nodes With `cmsh` And Bright View Add Functions .......... 252  
5.7.2 Adding New Nodes With The Node Creation Wizard ......................... 252  
5.8 Troubleshooting The Node Boot Process ............................................... 253  
5.8.1 Node Fails To PXE Boot .................................................................... 254  
5.8.2 Node-installer Logging ...................................................................... 257  
5.8.3 Provisioning Logging ......................................................................... 258  
5.8.4 Ramdisk Fails During Loading Or Sometime Later ............................... 258  
5.8.5 Ramdisk Cannot Start Network .......................................................... 258  
5.8.6 Node-Installer Cannot Create Disk Layout ....................................... 259  
5.8.7 Node-Installer Cannot Start BMC (IPMI/iLO) Interface ....................... 262  

6 User Management ...................................................................................... 267  
6.1 Managing Users And Groups With Bright View .................................... 267  
6.2 Managing Users And Groups With `cmsh` ............................................ 269  
6.2.1 Adding A User ................................................................................... 269  
6.2.2 Saving The Modified State ............................................................... 270  
6.2.3 Editing Properties Of Users And Groups ......................................... 271  
6.2.4 Reverting To The Unmodified State ................................................. 274  
6.2.5 Removing A User ............................................................................... 274  
6.3 Using An External LDAP Server ........................................................... 274  
6.3.1 External LDAP Server Replication ................................................... 274  
6.3.2 High Availability ............................................................................... 278  
6.4 Tokens And Profiles .............................................................................. 279  
6.4.1 Modifying Profiles .............................................................................. 281
### Table of Contents

6.4.2 Creation Of Custom Certificates With Profiles, For Users Managed By The Cluster Manager’s Internal LDAP ........................................ 281
6.4.3 Creation Of Custom Certificates With Profiles, For Users Managed By An External LDAP ......................................................... 284
6.4.4 Logging The Actions Of CMDaemon Users ........................................ 285

7 Workload Management .......................................................... 287

7.1 Workload Managers Choices .................................................. 287
7.2 Forcing Jobs To Run In A Workload Management System ............... 288
7.2.1 Disallowing User Logins To Regular Nodes Via cmsh ................... 288
7.2.2 Disallowing User Logins To Regular Nodes Via Bright View ........... 289
7.2.3 Disallowing Other User Processes Outside Of Workload Manager User Processes ......................... 290
7.2.4 High Availability By Workload Managers ................................... 290
7.3 Installation Of Workload Managers ............................................. 290
7.3.1 Running cm-wlm-setup In CLI Mode ......................................... 290
7.3.2 Running cm-wlm-setup In Ncurses Mode .................................... 294
7.3.3 Configuration Of Slurm To Run Containerized Jobs Using Pyxis With Enroot ...................... 295
7.3.4 Prolog And Epilog Scripts ...................................................... 297
7.4 Enabling, Disabling, And Monitoring Workload Managers .................. 301
7.4.1 Enabling And Disabling A WLM With Bright View ....................... 302
7.4.2 Enabling And Disabling A Workload Manager With cmsh ............... 305
7.4.3 Monitoring The Workload Manager Services ................................ 309
7.5 Configuring And Running Individual Workload Managers ................. 312
7.5.1 Configuring And Running Slurm .............................................. 313
7.5.2 Installing, Configuring, And Running UGE ................................. 323
7.5.3 Configuring And Running PBS ............................................... 329
7.5.4 Installing, Configuring, And Running LSF ................................. 334
7.6 Using Bright View With Workload Management .............................. 340
7.6.1 Jobs Display And Handling In Bright View .................................. 340
7.6.2 Queues Display And Handling In Bright View ............................... 341
7.7 Using cmsh With Workload Management ..................................... 342
7.7.1 The jobs Submode In cmsh .................................................... 344
7.7.2 Job Queue Display And Handling In cmsh: jobqueue Mode ............. 348
7.7.3 Nodes Drainage Status And Handling In cmsh ............................... 350
7.8 Examples Of Workload Management Assignment ............................ 352
7.8.1 Setting Up A New Category And A New Queue For It ..................... 352
7.8.2 Setting Up A Prejob Check .................................................... 356
7.9 Power Saving With cm-scale ........................................................................ 357
7.10 Cgroups ................................ .................................................... 358
7.10.1 Cgroups Settings For Workload Managers ................................... 358
7.11 Custom Node Parameters ....................................................... 365

8 NVIDIA Bright Cluster Manager Auto Scaler .................................. 367
8.1 Introduction ........................................................................... 367
8.1.1 Use Cases .......................................................................... 367
8.1.2 Resource Constraints .......................................................... 368
8.1.3 Setup ................................................................................. 371
<table>
<thead>
<tr>
<th>Section</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>8.1.4 Comparison With cm-scale-cluster</td>
<td>380</td>
</tr>
<tr>
<td>8.2 Configuration</td>
<td>381</td>
</tr>
<tr>
<td>8.2.1 The ScaleServer Role</td>
<td>381</td>
</tr>
<tr>
<td>8.2.2 Resource Providers</td>
<td>383</td>
</tr>
<tr>
<td>8.2.3 Time Quanta Optimization</td>
<td>386</td>
</tr>
<tr>
<td>8.2.4 Fairsharing Priority Calculation And Node Management</td>
<td>388</td>
</tr>
<tr>
<td>8.2.5 Engines</td>
<td>388</td>
</tr>
<tr>
<td>8.2.6 Trackers</td>
<td>390</td>
</tr>
<tr>
<td>8.3 Examples Of cm-scale Use</td>
<td>396</td>
</tr>
<tr>
<td>8.3.1 Simple Static Node Provider Usage Example</td>
<td>396</td>
</tr>
<tr>
<td>8.3.2 Simple Dynamic Node Provider Usage Example</td>
<td>399</td>
</tr>
<tr>
<td>8.4 Further cm-scale Configuration And Examples</td>
<td>405</td>
</tr>
<tr>
<td>8.4.1 Dynamic Nodes Re-purposing</td>
<td>405</td>
</tr>
<tr>
<td>8.4.2 Pending Reasons</td>
<td>406</td>
</tr>
<tr>
<td>8.4.3 Locations</td>
<td>407</td>
</tr>
<tr>
<td>8.4.4 Azure Storage Accounts Assignment</td>
<td>409</td>
</tr>
<tr>
<td>8.4.5 Mapping HPC Jobs To Particular Nodes</td>
<td>409</td>
</tr>
<tr>
<td>8.4.6 How To Exclude Unused Nodes From Being Stopped</td>
<td>411</td>
</tr>
<tr>
<td>8.4.7 Prolog And Epilog Scripts With Auto Scaler</td>
<td>411</td>
</tr>
<tr>
<td>8.4.8 Queue Node Placeholders</td>
<td>412</td>
</tr>
<tr>
<td>9 Containerization</td>
<td>415</td>
</tr>
<tr>
<td>9.1 Docker Engine</td>
<td>415</td>
</tr>
<tr>
<td>9.1.1 Docker Setup</td>
<td>416</td>
</tr>
<tr>
<td>9.1.2 Integration With Workload Managers</td>
<td>418</td>
</tr>
<tr>
<td>9.1.3 DockerHost Role</td>
<td>418</td>
</tr>
<tr>
<td>9.1.4 Iptables</td>
<td>420</td>
</tr>
<tr>
<td>9.1.5 Storage Backends</td>
<td>421</td>
</tr>
<tr>
<td>9.1.6 Docker Monitoring</td>
<td>424</td>
</tr>
<tr>
<td>9.1.7 Docker Setup For NVIDIA</td>
<td>425</td>
</tr>
<tr>
<td>9.2 Docker Registries</td>
<td>426</td>
</tr>
<tr>
<td>9.2.1 Docker And Harbor Registries: Introduction</td>
<td>426</td>
</tr>
<tr>
<td>9.2.2 Docker And Harbor Registries: Setup And Configuration</td>
<td>427</td>
</tr>
<tr>
<td>9.3 Kubernetes</td>
<td>429</td>
</tr>
<tr>
<td>9.3.1 Reference Architecture</td>
<td>429</td>
</tr>
<tr>
<td>9.3.2 Kubernetes Setup</td>
<td>430</td>
</tr>
<tr>
<td>9.3.3 Using GPUs With Kubernetes: NVIDIA GPUs</td>
<td>438</td>
</tr>
<tr>
<td>9.3.4 Using GPUs With Kubernetes: AMD GPUs</td>
<td>444</td>
</tr>
<tr>
<td>9.3.5 Kubernetes Configuration Overlays</td>
<td>448</td>
</tr>
<tr>
<td>9.3.6 Removing A Kubernetes Cluster</td>
<td>448</td>
</tr>
<tr>
<td>9.3.7 Kubernetes Cluster Configuration Options</td>
<td>449</td>
</tr>
<tr>
<td>9.3.8 EtcdCluster</td>
<td>452</td>
</tr>
<tr>
<td>9.3.9 Kubernetes Roles</td>
<td>453</td>
</tr>
<tr>
<td>9.3.10 Security Model</td>
<td>463</td>
</tr>
<tr>
<td>9.3.11 Addition Of New Kubernetes Users</td>
<td>464</td>
</tr>
<tr>
<td>9.3.12 Getting Information And Modifying Existing Kubernetes Users</td>
<td>466</td>
</tr>
<tr>
<td>9.3.13 List Of Resources Defined For Users</td>
<td>466</td>
</tr>
<tr>
<td>Section</td>
<td>Page</td>
</tr>
<tr>
<td>------------------------------------------------------------------------</td>
<td>------</td>
</tr>
<tr>
<td>9.3.14 Pod Security Policies</td>
<td>467</td>
</tr>
<tr>
<td>9.3.15 Kubernetes Permission Manager</td>
<td>471</td>
</tr>
<tr>
<td>9.3.16 Providing Access To External Users</td>
<td>474</td>
</tr>
<tr>
<td>9.3.17 Networking Model</td>
<td>476</td>
</tr>
<tr>
<td>9.3.18 Kubernetes Monitoring</td>
<td>476</td>
</tr>
<tr>
<td>9.3.19 Local Path Storage Class</td>
<td>477</td>
</tr>
<tr>
<td>9.3.20 Setup Of A Storage Class For Ceph</td>
<td>477</td>
</tr>
<tr>
<td>9.3.21 Integration With Harbor</td>
<td>479</td>
</tr>
<tr>
<td>9.4 Kubernetes Apps</td>
<td>480</td>
</tr>
<tr>
<td>9.4.1 Providing Custom Docker Images</td>
<td>481</td>
</tr>
<tr>
<td>9.5 Kubernetes Operators</td>
<td>481</td>
</tr>
<tr>
<td>9.6 Kubernetes On Edge</td>
<td>483</td>
</tr>
<tr>
<td>9.6.1 Flags For Edge Installation</td>
<td>483</td>
</tr>
<tr>
<td>9.7 Singularity</td>
<td>484</td>
</tr>
<tr>
<td>9.7.1 Use Cases</td>
<td>484</td>
</tr>
<tr>
<td>9.7.2 Package cm-singularity</td>
<td>484</td>
</tr>
<tr>
<td>9.7.3 MPI Integration</td>
<td>485</td>
</tr>
<tr>
<td>9.8 OpenShift Container Platform Integration With The Cluster Manager</td>
<td>486</td>
</tr>
<tr>
<td>9.8.1 Prerequisites</td>
<td>486</td>
</tr>
<tr>
<td>9.8.2 Installation</td>
<td>486</td>
</tr>
<tr>
<td>9.8.3 Adding New Compute Nodes</td>
<td>488</td>
</tr>
<tr>
<td>9.8.4 Validation</td>
<td>488</td>
</tr>
<tr>
<td>9.8.5 Uninstall</td>
<td>488</td>
</tr>
<tr>
<td><strong>10 Ceph Installation</strong></td>
<td>491</td>
</tr>
<tr>
<td>10.1 Ceph Introduction</td>
<td>491</td>
</tr>
<tr>
<td>10.1.1 Ceph Object And Block Storage</td>
<td>491</td>
</tr>
<tr>
<td>10.1.2 Ceph Storage Backends</td>
<td>492</td>
</tr>
<tr>
<td>10.1.3 Ceph Software Considerations Before Use</td>
<td>493</td>
</tr>
<tr>
<td>10.1.4 Hardware For Ceph Use</td>
<td>493</td>
</tr>
<tr>
<td>10.2 Ceph Installation With cm-ceph-setup</td>
<td>494</td>
</tr>
<tr>
<td>10.2.1 Ceph Installation: The Configuration Stage</td>
<td>495</td>
</tr>
<tr>
<td>10.2.2 Ceph Installation: The Deployment Stage</td>
<td>498</td>
</tr>
<tr>
<td>10.3 Installation Of Ceph From Bright View</td>
<td>500</td>
</tr>
<tr>
<td>10.3.1 Bright View Ceph Install: Main Details Screen</td>
<td>500</td>
</tr>
<tr>
<td>10.3.2 Bright View Ceph Install: Nodes Selection Screen</td>
<td>501</td>
</tr>
<tr>
<td>10.3.3 Bright View Ceph Install: Summary Screen</td>
<td>503</td>
</tr>
<tr>
<td>10.3.4 Bright View Ceph Install: Deployment Screen</td>
<td>503</td>
</tr>
<tr>
<td>10.4 Checking And Getting Familiar With Ceph Items After cm-ceph-setup</td>
<td>504</td>
</tr>
<tr>
<td>10.4.1 Checking On Ceph And Ceph-related Files From The Shell</td>
<td>504</td>
</tr>
<tr>
<td>10.4.2 Ceph Management With Bright View And cmsh</td>
<td>507</td>
</tr>
<tr>
<td>10.5 RADOS GW Installation, Initialization, And Properties</td>
<td>512</td>
</tr>
<tr>
<td>10.5.1 RADOS GW Installation And Initialization</td>
<td>512</td>
</tr>
<tr>
<td>10.5.2 Setting RADOS GW Properties</td>
<td>512</td>
</tr>
</tbody>
</table>
# 11 BeeGFS

11.1 BeeGFS Introduction .......................... 513
   11.1.1 BeeGFS Concepts .......................... 513
   11.1.2 BeeGFS Installation Notes And Options .... 513
11.2 Deployment And Uninstallation Of BeeGFS With `cm-beegfs-setup` .............. 513
   11.2.1 Deployment Of BeeGFS ......................... 514
   11.2.2 Uninstalling BeeGFS ......................... 518
11.3 Managing The Deployed BeeGFS Instance .............................................. 518
   11.3.1 Setup ........................................ 519
   11.3.2 BeeGFS Objects .............................. 519
   11.3.3 Usage ....................................... 533

# 12 Post-Installation Software Management

12.1 NVIDIA Bright Cluster Manager Packages And Their Naming Convention ....... 535
   12.1.1 The `packages` Command ........................................ 537
12.2 Managing Packages On The Head Node .............................................. 538
   12.2.1 Managing RPM Or .deb Packages On The Head Node .................... 538
   12.2.2 Installation Of Packages On The Head Node That Are Not .deb And Not .rpm Packages .............................................. 541
12.3 Kernel Management On A Head Node Or Image ........................................ 541
   12.3.1 Installing A Standard Distribution Kernel Into An Image Or On A Head Node .............. 541
   12.3.2 Excluding Kernels And Other Packages From Updates .............. 542
   12.3.3 Updating A Kernel In A Software Image .............................................. 543
   12.3.4 Setting Kernel Options For Software Images .............................. 544
   12.3.5 Kernel Driver Modules .............................................. 545
12.4 Managing A Package In A Software Image And Running It On Nodes .............. 547
   12.4.1 Installing From Head Into The Image: Changing The Root Directory Into Which The Packages Are Deployed .............. 547
   12.4.2 Installing From Head Into The Image: Updating The Node .............. 549
   12.4.3 Installing From Head Into The Image: Possible Issues When Using `rpm --root`, `yum --installroot` Or `chroot` .............................................. 549
   12.4.4 Managing A Package In The Node-Installer Image ......................... 550
12.5 Managing Non-RPM Software In A Software Image And Running It On Nodes .............. 551
   12.5.1 Managing The Software Directly On An Image .............................. 551
   12.5.2 Managing The Software Directly On A Node, Then Syncing Node-To-Image .............. 552
12.6 Creating A Custom Software Image .............................................. 555
   12.6.1 Creating A Base Distribution Archive From A Base Host .............. 555
   12.6.2 Creating The Software Image With `cm-create-image` .................. 557
   12.6.3 Configuring Local Repositories For Linux Distributions, And For The Cluster Manager Package Repository, For A Software Image .............. 562
   12.6.4 Creating A Custom Image From The Local Repository .............. 564
12.7 Creating Images For Other Distributions And Architectures (Multidistro And Multiarch) .............. 564
   12.7.1 The `cm-image` Tool .............................................. 565
   12.7.2 Multidistro Examples: Provisioning From CentOS 7 Head Node To Ubuntu 18.04 Regular Nodes .............................................. 568
   12.7.3 Multiarch Example: Creating An Image From A Centos 8 Head Node For ARMv8 Architecture Regular Nodes .............................................. 569
13 Monitoring: Monitoring Cluster Devices ........................................ 573
  13.1 A Basic Monitoring Example And Action ..................................... 573
    13.1.1 Synopsis Of Basic Monitoring Example .................................. 573
    13.1.2 Before Using The Basic Monitoring Example—Setting Up The Pieces .... 574
    13.1.3 Using The Basic Monitoring Example ...................................... 575
  13.2 Monitoring Concepts And Definitions ........................................ 578
    13.2.1 Measurables ................................................................. 578
    13.2.2 Enummetrics ............................................................... 580
    13.2.3 Metrics ..................................................................... 581
    13.2.4 Health Check ............................................................ 582
    13.2.5 Trigger ............................................................... 584
    13.2.6 Action ................................................................. 584
    13.2.7 Severity ................................................................. 585
    13.2.8 AlertLevel .............................................................. 585
    13.2.9 Flapping ................................................................. 585
    13.2.10 Data Producer .......................................................... 585
    13.2.11 Conceptual Overview: The Main Monitoring Interfaces Of Bright View .... 589
  13.3 Monitoring Visualization With Bright View .................................... 590
    13.3.1 The Monitoring Window .................................................. 590
  13.4 Monitoring Configuration With Bright View .................................... 592
    13.4.1 Monitoring Configuration: Data Producers .................................. 593
    13.4.2 Monitoring Configuration: Measurables ................................... 595
    13.4.3 Monitoring Configuration: Consolidators ................................... 597
    13.4.4 Monitoring Configuration: Actions ....................................... 600
    13.4.5 Monitoring Configuration: Triggers ....................................... 602
    13.4.6 Monitoring Configuration: Health status .................................. 605
    13.4.7 Monitoring Configuration: All Health Checks .............................. 606
    13.4.8 Monitoring Configuration: Standalone Monitored Entities ................. 607
  13.5 The monitoring Mode Of cmsh ................................................. 607
    13.5.1 The action Submode ...................................................... 608
    13.5.2 The consolidator Submode ................................................ 610
    13.5.3 The measurable Submode ................................................. 612
    13.5.4 The setup Submode ...................................................... 616
    13.5.5 The standalone Submode ................................................ 622
    13.5.6 The trigger Submode ..................................................... 622
  13.6 Obtaining Monitoring Data Values .............................................. 625
    13.6.1 Getting The List Of Measurables For An Entity: The measurables, metrics,
                   healthchecks And enummetrics Commands ................................ 625
    13.6.2 On-Demand Metric Sampling And Health Checks .............................. 626
    13.6.3 The Latest Data And Counter Values—The latest*data And
                   latestmetriccounters Commands ........................................ 629
    13.6.4 Data Values Over A Period—The dumpmonitoringdata Command .............. 631
    13.6.5 Monitoring Data Health Overview—The healthoverview Command ............. 639
    13.6.6 Monitoring Data About The Monitoring System—The monitoringinfo Command .... 639
    13.6.7 Dropping Monitoring Data With The monitoringdrop Command ............... 640
| 13.6.8 Monitoring Suspension And Resumption—The `monitoringsuspend` And `monitoringresume` Commands | 642 |
| 13.6.9 Monitoring Pickup Intervals | 643 |
| 13.7 Offloaded Monitoring | 645 |
| 13.7.1 Why Offloaded Monitoring? | 645 |
| 13.7.2 Implementing Offloaded Monitoring | 646 |
| 13.7.3 Background Details | 646 |
| 13.8 The User Portal | 650 |
| 13.8.1 Accessing The User Portal | 650 |
| 13.8.2 Setting A Common Username/Password For The User Portal | 651 |
| 13.8.3 User Portal Access | 651 |
| 13.8.4 User Portal Home Page | 652 |
| 13.9 Cloud Job Tagging | 653 |
| 13.10 Event Viewer | 653 |
| 13.10.1 Viewing Events In Bright View | 654 |
| 13.10.2 Viewing Events In `cmsh` | 654 |
| 13.10.3 Using The Event Bucket From The Shell For Events And For Tagging Device States | 656 |
| 13.10.4 InfoMessages | 656 |
| 13.11 Monitoring Location With GNSS | 658 |
| 13.12 Monitoring Report Queries | 658 |
| 13.12.1 Monitoring Report Queries In `cmsh` | 658 |
| 14 Monitoring: Job Monitoring | 661 |
| 14.1 Job Metrics Introduction | 661 |
| 14.2 Job Metrics With Cgroups | 661 |
| 14.3 Job Information Retention | 662 |
| 14.4 Job Metrics Sampling Configuration | 663 |
| 14.4.1 The Job Metrics Collection Processing Mechanism | 664 |
| 14.5 Job Monitoring In `cmsh` | 665 |
| 15 Monitoring: Job Accounting | 669 |
| 15.1 Introduction | 669 |
| 15.2 Labeled Entities | 669 |
| 15.2.1 Dataproducers For Labeled Entities | 669 |
| 15.2.2 PromQL And Labeled Entities | 670 |
| 15.2.3 Job IDs And Labeled Entities | 670 |
| 15.2.4 Measurables And Labeled Entities | 670 |
| 15.3 PromQL Queries | 671 |
| 15.3.1 The Default PromQL Queries... | 671 |
| 15.3.2 ...And A Short Description Of Them | 672 |
| 15.3.3 Modifying The Default PromQL Query Properties | 676 |
| 15.3.4 An Example PromQL Query, Properties, And Disassembly | 677 |
| 15.3.5 Aside: Getting Raw Values For A Prometheus Class Metric | 678 |
| 15.3.6 ...An Example PromQL Query, Properties, And Disassembly (Continued) | 679 |
| 15.4 Parameterized PromQL Queries | 680 |
| 15.5 Job Accounting In Bright View | 681 |
| 15.5.1 Management And Use Of The Accounting Panel | 681 |
## Table of Contents

15.6 PromQL Query Modes And Specification In Bright View .......................... 684
15.7 Access Control For Workload Accounting And Reporting ......................... 686
  15.7.1 Defining Project Managers Using Internal User Management .......... 687
  15.7.2 Defining Project Managers Using External User Management .......... 687
15.8 Drilldown Queries For Workload Accounting And Reporting ....................... 688
  15.8.1 The `drilldownoverview` Command ....................................... 689

16 Monitoring: Job Chargeback .................................................. 693
  16.1 Introduction ................................................................. 693
    16.1.1 The Word “Chargeback” ............................................. 693
    16.1.2 Comparison Of Job Chargeback Monitoring Measurement With Other Monitoring Measurements ..................... 693
  16.2 Job Chargeback Measurement ............................................... 694
    16.2.1 Predefined Job Chargebacks ......................................... 694
    16.2.2 Setting A Custom Job Chargeback .................................... 695
    16.2.3 The `report` And `request` Commands ................................ 696

17 Day-to-day Administration ...................................................... 700
  17.1 Parallel Shells: `pdsh` And `pexec` ...................................... 701
    17.1.1 `pdsh` In The OS Shell .............................................. 702
    17.1.2 `pexec` In `cmsh` ................................................... 705
    17.1.3 `pexec` In Bright View .............................................. 705
    17.1.4 Using The `-j|--join` Option Of `pexec` In `cmsh` ............... 706
    17.1.5 Other Parallel Commands ............................................ 707
  17.2 Getting Support With Cluster Manager Issues, And Notifications For Release Updates . 707
    17.2.1 Support Via E-mail ................................................... 707
    17.2.2 Reporting Cluster Manager Diagnostics With `cm-diagnose` .......... 708
    17.2.3 Requesting Remote Support With `request-remote-assistance` ...... 709
    17.2.4 Requesting Remote Support With A Shared Screen Utility .......... 711
    17.2.5 Getting Notified About Updates ..................................... 711
  17.3 Backups ................................................................. 712
    17.3.1 Cluster Installation Backup ........................................ 712
    17.3.2 Local Database And Data Backups And Restoration ................. 713
  17.4 Revision Control For Images ............................................. 715
    17.4.1 Btrfs: The Concept And Why It Works Well In Revision Control For Images ........................................... 715
    17.4.2 Btrfs Availability And Distribution Support ......................... 715
    17.4.3 Installing Btrfs To Work With Revision Control Of Images In The Cluster Manager ........................... 716
    17.4.4 Using `cm` For Revision Control Of Images ......................... 718
  17.5 BIOS Configuration And Updates ......................................... 720
    17.5.1 BIOS Configuration Via `CMDaemon` And `Redfish` ................. 720
    17.5.2 Updating BIOS Versions ............................................... 727
  17.6 Hardware Match Check With The `hardware-profile` Data Producer .......... 729
  17.7 Serial Over LAN Console Access ........................................ 730
    17.7.1 Background Notes On Serial Console And `SOL` ..................... 731
    17.7.2 `SOL` Console Configuration With Bright View ..................... 732
    17.7.3 `SOL` Console Configuration And Access With `cmsh` ............... 732
17.7.4 The `conman` Serial Console Logger And Viewer ........................................... 733
17.8 Managing Raw Monitoring Data ................................................................. 737
  17.8.1 Monitoring Subsystem Disk Usage With The `monitoringinfo --storage` Option 737
  17.8.2 Estimating The Required Size Of The Storage Device .............................. 737
  17.8.3 Moving Monitoring Data Elsewhere ....................................................... 738
  17.8.4 Reducing Monitoring Data By Reducing Samples .................................. 738
  17.8.5 Deleting All Monitoring Data .............................................................. 738
17.9 Node Replacement ......................................................................................... 739
17.10 Ansible And NVIDIA Bright Cluster Manager ............................................. 740
    17.10.1 An Overview Of Ansible ..................................................................... 740
    17.10.2 A Simple Playbook Example ............................................................. 741
    17.10.3 An Intermediate Playbook Example: Setting Up A Cluster For Demonstration Purposes ................................................................. 743
    17.10.4 A More Complicated Playbook Example: Creating An Edge Site And Related Properties ................................................................. 746
18 High Availability ............................................................................................ 753
  18.0 Introduction ................................................................................................. 753
    18.0.1 Why Have High Availability? ............................................................... 753
    18.0.2 High Availability—For What Nodes? ................................................. 753
    18.0.3 High Availability Usually Uses Shared Storage .................................... 754
    18.0.4 Organization Of This Chapter .............................................................. 754
  18.1 HA Concepts ............................................................................................... 754
    18.1.1 Primary, Secondary, Active, Passive ................................................. 754
    18.1.2 Monitoring The Active Head Node, Initiating Failover ......................... 754
    18.1.3 Services In The Cluster Manager HA Setups ....................................... 755
    18.1.4 Failover Network Topology ................................................................ 756
    18.1.5 Shared Storage ................................................................................. 757
    18.1.6 Guaranteeing One Active Head At All Times ....................................... 758
    18.1.7 Automatic Vs Manual Failover ............................................................ 759
    18.1.8 HA And Cloud Nodes ....................................................................... 760
    18.1.9 HA Using Virtual Head Nodes ........................................................... 760
  18.2 HA Setup Procedure Using `cmha-setup` .................................................... 760
    18.2.1 Preparation ....................................................................................... 761
    18.2.2 Failover Cloning (Replacing A Passive Head) ...................................... 763
    18.2.3 Shared Storage Setup ....................................................................... 766
    18.2.4 Automated Failover And Relevant Testing ......................................... 768
  18.3 Running `cmha-setup` Without Ncurses, Using An XML Specification ......... 769
    18.3.1 Why Run It Without Ncurses? ............................................................ 769
    18.3.2 The Syntax Of `cmha-setup` Without Ncurses .................................... 769
    18.3.3 Example `cmha-setup` Run Without Ncurses ...................................... 770
  18.4 Managing HA .............................................................................................. 770
    18.4.1 Changing An Existing Failover Configuration ..................................... 770
    18.4.2 `cmha` Utility .................................................................................... 771
    18.4.3 States ............................................................................................... 774
    18.4.4 Failover Action Decisions ................................................................. 775
    18.4.5 Keeping Head Nodes In Sync .............................................................. 776
18.4.6 High Availability Parameters ........................................ 778
18.4.7 Viewing Failover Via Bright View .................................... 780
18.4.8 Re-cloning A Head Node ............................................. 780
18.5 HA For Regular Nodes And Edge Director Nodes ...................... 781
  18.5.1 Why Have HA On Non-Head Nodes? ................................. 781
  18.5.2 Comparing HA For Head Nodes, Regular Nodes And Edge Director Nodes ................. 781
  18.5.3 Setting Up A Regular Node HA Service ................................ 782
  18.5.4 The Sequence Of Events When Making Another HA Regular Node Active ............... 786
18.6 HA And Workload Manager Jobs ........................................ 786

19 The Jupyter Notebook Environment Integration ............................ 789
  19.1 Introduction ........................................................... 789
  19.2 Jupyter Environment Installation ....................................... 790
    19.2.1 Jupyter Setup .................................................. 791
    19.2.2 Jupyter Architecture ........................................... 791
    19.2.3 Verifying Jupyter Installation ................................... 793
  19.3 Jupyter Notebook Examples ........................................... 795
  19.4 Jupyter Kernels ..................................................... 796
    19.4.1 Jupyter Enterprise Gateway Kernels ............................ 797
  19.5 Jupyter Kernel Creator Extension ...................................... 798
    19.5.1 Running Jupyter Kernels With Kubernetes ....................... 803
    19.5.2 Running Jupyter Kernels With Workload Managers ............... 804
  19.6 Jupyter Kernel Creator Extension Customization ...................... 805
    19.6.1 Kernel Template Parameters Definition ......................... 805
    19.6.2 Kernel Template Parameters Usage ................................ 808
    19.6.3 Kernel Template Creation Example ................................ 808
    19.6.4 Extending Kubernetes Kernel Templates ........................... 810
    19.6.5 Extending Workload Managers Kernel Templates .................. 812
    19.6.6 Filtering Out Irrelevant Templates From The Interface For Users .................. 814
  19.7 Kubernetes Jupyter Kernel Operator ................................... 815
    19.7.1 Installing Jupyter Kernel Operator ............................... 815
    19.7.2 Architecture Overview .......................................... 816
    19.7.3 Running Jupyter Kernel Using The Operator ...................... 818
    19.7.4 Kernel Operator Tunables ....................................... 819
    19.7.5 Sidecar Arguments And Environment Variables ................. 820
    19.7.6 Running Spark-based Kernels In Jupyter Kernel Operator ....... 821
    19.7.7 Example: Creating An R kernel .................................. 821
  19.8 Jupyter VNC Extension ................................................ 826
  19.9 Jupyter WLM Magic Extension ......................................... 828
  19.10 Jupyter Environment Removal ......................................... 831

A Generated Files .......................................................... 833
  A.1 System Configuration Files Created Or Modified By CMDeamon On Head Nodes ................ 833
  A.2 System Configuration Files Created Or Modified By CMDaemon In Software Images: .......... 835
  A.3 Files Created On Regular Nodes By The Node-Installer .................. 836
  A.4 Files Not Generated, But Installed In RHEL And Derivatives ...................... 837
# Table of Contents

B Bright Computing Public Key 843

C CMDaemon Configuration File Directives 845

D Disk Partitioning And RAID Configuration 875

D.1 Structure Of Partitioning Definition—The Global Partitioning XML Schema File 875
D.2 Structure Of Hardware RAID Definition—The Hardware RAID XML Schema File 882
D.3 Example: Default Node Partitioning 885
D.4 Example: Hardware RAID Configuration

D.4.1 RAID level 0 And RAID 10 Example 887
D.5 Example: Software RAID 889
D.6 Example: Software RAID With Swap 890
D.7 Example: Logical Volume Manager 891
D.8 Example: Logical Volume Manager With RAID 1 892
D.9 Example: Diskless 893
D.10 Example: Semi-diskless 894
D.11 Example: Preventing Accidental Data Loss 894
D.12 Example: Using Custom Assertions 896
D.13 Example: Software RAID1 With One Big Partition 897
D.14 Example: Software RAID5 With One Big Partition 898
D.15 Example: Software RAID1 With Standard Partitioning 900
D.16 Example: Software RAID5 With Standard Partitioning 903
D.17 Example: LUKS Disk Encryption With Standard Partitioning 906

D.17.1 Introduction 906
D.17.2 Node Provisioned Over The Network: Encrypted Partition XML Example 906
D.17.3 Standalone Node: Encrypted Partition XML Example 909
D.17.4 Changing A Passphrase On An Encrypted Node 910

E Example initialize And finalize Scripts 911

E.1 When Are They Used? 911
E.2 Accessing From Bright View And cmsh 911
E.3 Environment Variables Available To initialize And finalize Scripts 912
E.4 Using Environment Variables Stored In Multiple Variables 915
E.5 Storing A Configuration To A Filesystem

E.5.1 Storing With initialize Scripts 916
E.5.2 Ways Of Writing A finalize Script To Configure The Destination Nodes 916
E.5.3 Restricting The Script To Nodes Or Node Categories 919

F Workload Managers Quick Reference 921

F.1 Slurm 921
F.2 Univa Grid Engine 923
F.3 PBS Pro 924

G Metrics, Health Checks, Enummetrics, And Actions 925

G.1 Metrics And Their Parameters 925
G.1.1 Regular Metrics 926
G.1.2 NFS Metrics 932
G.1.3 InfiniBand Metrics 933
Preface

Welcome to the Administrator Manual for the NVIDIA Bright 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 NVIDIA Bright 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 NVIDIA Bright 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.brightcomputing.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.

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 Bright Computing. So any comments, suggestions or corrections will be very gratefully accepted at manuals@brightcomputing.com.

There is also a feedback form available via Bright View, via the Menu icon, following the click-path: 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 17.2 has more details on working with support.

0.5 Getting Professional Services

Bright Computing normally differentiates between

- professional services (customer asks Bright Computing to do something or asks Bright Computing 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 Bright account manager.
1

Introduction

1.1 NVIDIA Bright Cluster Manager Functions And Aims

NVIDIA Bright Cluster Manager contains tools and applications to facilitate the installation, administra-
tion, 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 differ-
ence 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 12 of this manual.

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

• **Cloudbursting, Edge, and Machine Learning:** these are integrated by Bright Computing in vari-
ous 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 (Bright 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:

<table>
<thead>
<tr>
<th>Chapter, Title</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>2 Cluster Management With NVIDIA Bright Cluster Manager</td>
<td>Introduction to main concepts and tools of the cluster manager. Lays down groundwork for the remaining chapters</td>
</tr>
<tr>
<td>3 Configuring The Cluster</td>
<td>Further configuration and set up of the cluster after software installation of the cluster manager on the head node.</td>
</tr>
<tr>
<td>4 Power Management</td>
<td>How power management within the cluster works</td>
</tr>
<tr>
<td>5 Node Provisioning</td>
<td>Node provisioning in detail</td>
</tr>
<tr>
<td>6 User Management</td>
<td>Account management for users and groups</td>
</tr>
<tr>
<td>7 Workload Management</td>
<td>Workload management implementation and use</td>
</tr>
<tr>
<td>8 The cm-scale Service</td>
<td>A cluster manager service to dynamically scale the cluster according to need</td>
</tr>
<tr>
<td>9 Containerization</td>
<td>Using Docker and Kubernetes with the cluster manager</td>
</tr>
<tr>
<td>10 Ceph Installation</td>
<td>Installing Ceph scalable storage</td>
</tr>
<tr>
<td>11 BeeGFS Installation</td>
<td>Installing BeeGFS high performance storage</td>
</tr>
<tr>
<td>12 Post-Installation Software Management</td>
<td>Managing, updating, modifying the cluster manager software and images</td>
</tr>
<tr>
<td>13 Monitoring: Monitoring Cluster Devices</td>
<td>Device monitoring and conditional action triggers</td>
</tr>
<tr>
<td>14 Monitoring: Job Monitoring</td>
<td>Jobs resource consumption monitoring by the jobs</td>
</tr>
<tr>
<td>15 Monitoring: Job Accounting</td>
<td>Jobs resource consumption monitoring aggregated by user or similar groupings</td>
</tr>
<tr>
<td>16 Monitoring: Job Charge-back</td>
<td>Resource request monitoring, so that groups of users can be charged for their use</td>
</tr>
<tr>
<td>17 Day-To-Day Administration</td>
<td>Miscellaneous administration</td>
</tr>
<tr>
<td>18 High Availability</td>
<td>Background details and setup instructions to build a cluster with redundant head nodes</td>
</tr>
<tr>
<td>19 The Jupyter Notebook Environment Integration</td>
<td>Installing and using the Jupyter notebook environment</td>
</tr>
</tbody>
</table>

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)

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 17.2 describes how to get further support.
This chapter introduces cluster management with NVIDIA Bright 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 (Bright 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:

<table>
<thead>
<tr>
<th>device statuses</th>
<th>device is</th>
<th>monitored by Bright?</th>
<th>state tracking?</th>
</tr>
</thead>
<tbody>
<tr>
<td>[ UP ]</td>
<td>UP</td>
<td>monitored</td>
<td>tracked</td>
</tr>
<tr>
<td>[ DOWN ]</td>
<td>DOWN</td>
<td>monitored</td>
<td>tracked</td>
</tr>
<tr>
<td>[ CLOSED ] (UP)</td>
<td>UP</td>
<td>mostly ignored</td>
<td>tracked</td>
</tr>
<tr>
<td>[ CLOSED ] (DOWN)</td>
<td>DOWN</td>
<td>mostly ignored</td>
<td>tracked</td>
</tr>
</tbody>
</table>

These, and other 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 12.

### 2.1.3 Node 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 Bright 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 76), 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.

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.

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

<table>
<thead>
<tr>
<th>Node Category</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>nodes-ib</td>
<td>nodes with InfiniBand capabilities</td>
</tr>
<tr>
<td>nodes-highmem</td>
<td>nodes with extra memory</td>
</tr>
<tr>
<td>login</td>
<td>login nodes</td>
</tr>
<tr>
<td>storage</td>
<td>storage nodes</td>
</tr>
</tbody>
</table>

### 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:
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@bright92 ~]$ 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@bright92 ~]# 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@bright92 ~]# 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-environment 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@bright92 ~]# 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 com-
mands.

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

On a Bright 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 clusterwide can be
done by setting an environment variable, $ENABLE_LMOD in the cm-lmod-init.sh shell script:

Example

[root@bright92 ~]# 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@bright92 ~]# 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 made in the file is effective after having logged out, then logged into the shell again.
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@bright92 ~]# 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:
```
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 I.

If the administrator has stored the password to the cluster in the Bright View front-end, then the password should be modified there too (figure 2.1).

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

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

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 284.

Using Bright View and authenticating to the cluster manager: When an administrator uses the Bright View front end, a login to the cluster is carried out with username password authentication (figure 2.1), 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:

  ```bash
  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:

    ```bash
    about:preferences#privacy→Certificates→View Certificates→Your Certificates→Import
    ```

    The browser can then access the Bright 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 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 Bright View GUI

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

Bright 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 NVIDIA Bright Cluster Manager. For example, at the time of writing of this section, January 2021, the latest 2 versions were:

<table>
<thead>
<tr>
<th>Browser</th>
<th>Versions</th>
</tr>
</thead>
<tbody>
<tr>
<td>Chrome</td>
<td>86, 87</td>
</tr>
<tr>
<td>Edge</td>
<td>86, 87</td>
</tr>
<tr>
<td>Firefox</td>
<td>83, 84</td>
</tr>
<tr>
<td>Safari</td>
<td>13, 14</td>
</tr>
</tbody>
</table>

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

The GUI interface is provided by default as a web service on port 8081 from the head node to the browser. The URL takes the form:

https://<host name or IP address>:8081/bright-view

The cluster manager package that provides the service is bright-view, the web GUI is called Bright View and it is installed by default with the cluster manager. The service can be disabled by removing the package.

NVIDIA Bright Cluster Manager Bright View Login Window

Figure 2.1 shows the login dialog window for Bright View

![Bright View Login](https://<host name or IP address>:8081/bright-view)

NVIDIA Bright Cluster Manager Bright View Default Display On Connection

Clicking on the Login button logs the administrator into the Bright View service on the cluster. By default an overview window is displayed, corresponding to the path Cluster → Partition base (figure 2.2).
Figure 2.2: Cluster Overview

2.4.2 Navigating The Cluster With Bright View

Aspects of the cluster can be managed by administrators using Bright View (figure 2.2).

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.\footnote{The name Partition base is literally a footnote in the cluster manager history. It derives from the time that Bright 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.}

The tabs within the Partition base window are mapped out in figure 2.3 and summarily described next.
Figure 2.3: Cluster Navigation Within The Partition Base Window
Overview
The Overview tab (figure 2.2, page 16) shows the occupation rate (page 929), 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 NVIDIA Bright Cluster Manager version 7.0 and above is of a type that is compatible with versions all the way up to 9.2. NVIDIA Bright 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. Bright Computing 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, \texttt{cmsh}. This is the command-line interface to cluster management in the cluster manager. Since \texttt{cmsh} and Bright 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 Bright View may therefore safely skip this section.

The \texttt{cmsh} front end allows commands to be run with it, and can be used in batch mode. Although \texttt{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 \texttt{cmsh} in batch mode.

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

2.5.1 Invoking \texttt{cmsh}

From the head node, \texttt{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 \texttt{cmsh localhost}, then the management interface is most probably not up. In that case, bringing the management interface up allows \texttt{cmsh} to connect to CMDaemon.

Running \texttt{cmsh} without arguments starts an interactive cluster management session. To go back to the unix shell, a user enters \texttt{quit} or \texttt{ctrl-d}:

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

Batch Mode And Piping In \texttt{cmsh}

The \texttt{-c} flag allows \texttt{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 \texttt{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 \texttt{cmsh}

In a similar way to unix shells, \texttt{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:

```
[bright92]# 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
tdf -h
[root@mycluster ~]# cmsh
[mycluster->device]$ 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
| 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
| 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
/dev/vdb1 25G 17G 8.7G 66% /
tmpfs 374M 0 374M 0% /run/user/0

For NVIDIA Bright Cluster Manager 9.1 and higher, 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 (figure 2.5.1).

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
```
2.5 Cluster Management Shell

Echo all commands

--quit|-q
Exit immediately after error

--disable-multiline|-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

cmd
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 2.4 shows the top-level commands available in cmsh. These commands are displayed when help is typed in at the top-level of cmsh:
<table>
<thead>
<tr>
<th>Command</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>alias</td>
<td>Set aliases</td>
</tr>
<tr>
<td>category</td>
<td>Enter category mode</td>
</tr>
<tr>
<td>ceph</td>
<td>Enter ceph mode</td>
</tr>
<tr>
<td>cert</td>
<td>Enter cert mode</td>
</tr>
<tr>
<td>cloud</td>
<td>Enter cloud mode</td>
</tr>
<tr>
<td>cmjob</td>
<td>Enter cmjob mode</td>
</tr>
<tr>
<td>color</td>
<td>Manage console text color settings</td>
</tr>
<tr>
<td>configurationoverlay</td>
<td>Enter configurationoverlay mode</td>
</tr>
<tr>
<td>connect</td>
<td>Connect to cluster</td>
</tr>
<tr>
<td>delimiter</td>
<td>Display/set delimiter</td>
</tr>
<tr>
<td>device</td>
<td>Enter device mode</td>
</tr>
<tr>
<td>disconnect</td>
<td>Disconnect from cluster</td>
</tr>
<tr>
<td>edgesite</td>
<td>Enter edgesite mode</td>
</tr>
<tr>
<td>etcd</td>
<td>Enter etcd mode</td>
</tr>
<tr>
<td>events</td>
<td>Manage events</td>
</tr>
<tr>
<td>exit</td>
<td>Exit from current object or mode</td>
</tr>
<tr>
<td>export</td>
<td>Display list of aliases current list formats</td>
</tr>
<tr>
<td>fspart</td>
<td>Enter fspart mode</td>
</tr>
<tr>
<td>group</td>
<td>Enter group mode</td>
</tr>
<tr>
<td>groupingsyntax</td>
<td>Manage the default grouping syntax</td>
</tr>
<tr>
<td>help</td>
<td>Display this help</td>
</tr>
<tr>
<td>hierarchy</td>
<td>Enter hierarchy mode</td>
</tr>
<tr>
<td>history</td>
<td>Display command history</td>
</tr>
<tr>
<td>keyvaluestore</td>
<td>Enter keyvaluestore mode</td>
</tr>
<tr>
<td>kubernetes</td>
<td>Enter kubernetes mode</td>
</tr>
<tr>
<td>list</td>
<td>List state for all modes</td>
</tr>
<tr>
<td>main</td>
<td>Enter main mode</td>
</tr>
<tr>
<td>modified</td>
<td>List modified objects</td>
</tr>
<tr>
<td>monitoring</td>
<td>Enter monitoring mode</td>
</tr>
<tr>
<td>network</td>
<td>Enter network mode</td>
</tr>
<tr>
<td>nodegroup</td>
<td>Enter nodegroup mode</td>
</tr>
<tr>
<td>partition</td>
<td>Enter partition mode</td>
</tr>
<tr>
<td>process</td>
<td>Enter process mode</td>
</tr>
<tr>
<td>profile</td>
<td>Enter profile mode</td>
</tr>
<tr>
<td>quit</td>
<td>Quit shell</td>
</tr>
<tr>
<td>quitconfirmation</td>
<td>Manage the status of quit confirmation</td>
</tr>
<tr>
<td>rack</td>
<td>Enter rack mode</td>
</tr>
<tr>
<td>refresh</td>
<td>Refresh all modes</td>
</tr>
<tr>
<td>run</td>
<td>Execute cmsh commands from specified file</td>
</tr>
<tr>
<td>session</td>
<td>Enter session mode</td>
</tr>
<tr>
<td>softwareimage</td>
<td>Enter softwareimage mode</td>
</tr>
<tr>
<td>task</td>
<td>Enter task mode</td>
</tr>
<tr>
<td>time</td>
<td>Measure time of executing command</td>
</tr>
<tr>
<td>unalias</td>
<td>Unset aliases</td>
</tr>
<tr>
<td>unmanagednodeconfiguration</td>
<td>Enter unmanagednodeconfiguration mode</td>
</tr>
<tr>
<td>user</td>
<td>Enter user mode</td>
</tr>
<tr>
<td>watch</td>
<td>Execute a command periodically, showing output</td>
</tr>
<tr>
<td>wlm</td>
<td>Enter wlm mode</td>
</tr>
</tbody>
</table>

Figure 2.4: 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**
2.5 Cluster Management Shell

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

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

Options:
  -x, --echo
      Echo all commands

  -q, --quiet
      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 M.1.

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

• To enter a mode, a user enters the mode name at the cmsh prompt. The prompt changes to indicate that cmsh is in the requested mode, and commands for that mode can then be run.

• To use an object within a mode, the use command is used with the object name. In other words, a mode is entered, and an object within that mode is used. When an object is used, the prompt changes to indicate that that object within the mode is now being used, and that commands are applied for that particular object.

• To leave a mode, and go back up a level, the exit command is used. Similarly, if an object is in use, the exit command exits the object. At the top level, exit has the same effect as the quit command, that is, the user leaves cmsh and returns to the unix shell. The string .. is an alias for exit.

• The home command, which is aliased to /, takes the user from any mode depth to the top level.
The `path` command at any mode depth displays a string that can be used as a path to the current mode and object, in a form that is convenient for copying and pasting into `cmsh`. The string can be used in various ways. For example, it can be useful to define an alias in `.cmshrc` (page 20).

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**

```
[bright92->configurationoverlay[slurm-client]->roles[slurmclient]]% list
Name (key)
----------------------------
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:

```
[bright92 ~]# 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**

```
[bright92]% device
[bright92->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 bright92 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 ]
[bright92->device]% exit
[bright92]% device
[bright92->device]% use node001
[bright92->device[node001]]% path
home;device;use node001;
[bright92->device[node001]]% home
[bright92]% home;device;use node001 #copy-pasted from path output earlier
[bright92->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:
2.5 Cluster Management Shell

\[ \text{<mode> <cmd> <arg> . . . ; . . . ; <mode> <cmd> <arg> . . .} \]

where \(<mode>\) and \(<arg>\) are optional. ²

Example

\[
\text{[bright92->network]\% device status bright92; list}
\]

```
Name (key) Type Netmask bits Base address Domain name Ipv6
------------- --------- ------------- ------------- -------------------- ----
externalnet External 16 192.168.1.0 brightcomputing.com no
globalnet Global 0 0.0.0.0 cm.cluster
internalnet Internal 16 10.141.0.0 eth.cluster
```

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

\[
\text{[bright92->network]\% device; status bright92; list}
\]

```
Type Hostname (key) MAC Category Ip Network Status
------------- --------------- ------------------ --------- -------------- ----------- ----
HeadNode bright92 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 ]
```

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:

<table>
<thead>
<tr>
<th>Command</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>use</td>
<td>Use the specified object. I.e.: Make the specified object the current object</td>
</tr>
<tr>
<td>add</td>
<td>Create the object and use it</td>
</tr>
<tr>
<td>assign</td>
<td>Assign a new object</td>
</tr>
<tr>
<td>unassign</td>
<td>Unassign an object</td>
</tr>
<tr>
<td>clear</td>
<td>Clear the values of the object</td>
</tr>
<tr>
<td>clone</td>
<td>Clone the object and use it</td>
</tr>
<tr>
<td>remove</td>
<td>Remove the object</td>
</tr>
<tr>
<td>commit</td>
<td>Commit local changes, done to an object, to CMDaemon</td>
</tr>
</tbody>
</table>

²A more precise synopsis is:

\[ [\text{<mode>}] \text{ <cmd> [<arg> . . . ]] [: . . . ; [\text{<mode>}] \text{ <cmd> [<arg> . . . ]}] \]
Command | Description
--- | ---
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**

```bash
[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**

```bash
[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
```
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:

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 an 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%] remove node100
[mycluster->device%] commit
```

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 13.5.

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.
2.5 Cluster Management Shell

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, node01 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
[bright92->device[mycluster]->interfaces]% swap eth0 eth1; commit
[mycluster->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 Bright View when viewing Unsaved entities (figure 13.5). 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
  ```

  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**

  ```
  [bright92->device]% list -f hostname:14,switchports:15,ip
  hostname (key) switchports ip
  ----------------- --------------- ---------------------
  apc01 10.142.254.1
  bright92 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
  ```

  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

[bright92->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

[bright92->device]% format hostname:10
[bright92->device]% list
hostname (key)
----------
apc01
bright92
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

[bright92->device]% format hostname:[10-14]
[bright92->device]% list
hostname (key)
----------
apc01
bright92
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

[bright92->device]% format hostname:[10-14],switchports:14,ip:20
[bright92->device]% list
hostname (key) switchports ip
The output of the `format` command without arguments shows the current list printing format, with spaces. This can be used with enclosing quotes ("), or alternatively, with the spaces removed:

**Example**

```
[bright92->device]% format "hostname:[16-32], network:[14-32], status:[16-32]"
```

or

```
[bright92->device]% format hostname:[16-32],network:[14-32],status:[16-32]
```

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

**Example**

```
[bright92->device]% format -r
[bright92->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]
```

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**

```
[bright92->device]% sort type mac
```

```
[bright92->device]% list -f type:15,hostname:15,mac
type hostname (key) mac
--------------- --------------- --------------------
HeadNode bright92 08:0A:27:BA:B9:43
PhysicalNode node002 00:00:00:00:00:00
PhysicalNode log001 52:54:00:DE:E3:6B
```

```
[bright92->device]% sort hostname
```

```
[bright92->device]% list -f type:15,hostname:15,mac
type hostname (key) mac
--------------- --------------- --------------------
HeadNode bright92 08:0A:27:BA:B9:43
PhysicalNode log001 52:54:00:DE:E3:6B
PhysicalNode node002 00:00:00:00:00:00
```

```
[bright92->device]% sort mac hostname
```

```
[bright92->device]% list -f type:15,hostname:15,mac
type hostname (key) mac
--------------- --------------- --------------------
PhysicalNode node002 00:00:00:00:00:00
HeadNode bright92 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:

```bash
[bright92->device]% list -f type:15,hostname:15,mac --sort type,mac
[bright92->device]% list -f type:15,hostname:15,mac --sort type,hostname
[bright92->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**

```bash
[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 241) by using the close command.

**Example**

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

```bash
[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
```

<table>
<thead>
<tr>
<th>Code</th>
<th>Field</th>
<th>Message</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>category</td>
<td>The category should be set</td>
</tr>
</tbody>
</table>

```
[mycluster->device[node001]]# set category default
[mycluster->device[node001]]# validate
All good
[mycluster->device[node001]]# commit
```

**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
```

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>Activation</td>
<td>Thu, 03 Aug 2017 15:57:42 CEST</td>
</tr>
<tr>
<td>BMC Settings</td>
<td><code>&lt;submode&gt;</code></td>
</tr>
<tr>
<td>Block devices cleared on next boot</td>
<td>default</td>
</tr>
<tr>
<td>Category</td>
<td>default</td>
</tr>
<tr>
<td>...</td>
<td>...</td>
</tr>
<tr>
<td>Data node</td>
<td>no</td>
</tr>
<tr>
<td>Default gateway</td>
<td><code>10.141.255.254 (network: internalnet)</code></td>
</tr>
<tr>
<td>...</td>
<td>...</td>
</tr>
<tr>
<td>Software image</td>
<td><code>default-image</code></td>
</tr>
<tr>
<td>Static routes</td>
<td><code>&lt;0 in submode&gt;</code></td>
</tr>
</tbody>
</table>

**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` sub-mode, 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**

```bash
[root@bright92 ~]# cmsh
[bright92] handle roles node001
[bright92->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**

```bash
[bright92->device[node001]->roles]# import TAB TAB
cephmds cloudgateway elasticsearch...
[bright92->device[node001]->roles]# import --overlay slurm-client slurmclient
[bright92->device[node001]->roles]# list
Name (key)
-----------------------------------
[category:default] cgroupsupervisor
slurmclient
```

```bash
[bright92->device[node001]->roles]# set slurmclient queues node1q
[bright92->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 `node001` 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
```

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>Administrator e-mail</td>
<td></td>
</tr>
<tr>
<td>ArchOS</td>
<td>&lt;0 in submode&gt;</td>
</tr>
<tr>
<td>BMC Settings</td>
<td>&lt;submode&gt;</td>
</tr>
<tr>
<td>Burn configs</td>
<td>&lt;2 in submode&gt;</td>
</tr>
<tr>
<td>Cluster name</td>
<td>my-cluster</td>
</tr>
<tr>
<td>Default burn configuration</td>
<td>default-destructive</td>
</tr>
<tr>
<td>Default category</td>
<td>default</td>
</tr>
<tr>
<td>External network</td>
<td>externalnet</td>
</tr>
<tr>
<td>Externally visible IP</td>
<td></td>
</tr>
<tr>
<td>Failover</td>
<td>not defined</td>
</tr>
<tr>
<td>Failover groups</td>
<td>&lt;0 in submode&gt;</td>
</tr>
<tr>
<td>Headnode</td>
<td>my-headnode</td>
</tr>
<tr>
<td>Management network</td>
<td>internalnet</td>
</tr>
<tr>
<td>Name</td>
<td>base</td>
</tr>
<tr>
<td>Name servers</td>
<td>192.168.101.1</td>
</tr>
<tr>
<td>Name servers from dhcp</td>
<td>4.2.2.4</td>
</tr>
<tr>
<td>No zero conf</td>
<td>no</td>
</tr>
<tr>
<td>Node basename</td>
<td>node</td>
</tr>
<tr>
<td>Node digits</td>
<td>3</td>
</tr>
<tr>
<td>Notes</td>
<td>&lt;0 bytes&gt;</td>
</tr>
<tr>
<td>Provisioning Node Auto Update Timeout</td>
<td>300</td>
</tr>
<tr>
<td>Relay Host</td>
<td></td>
</tr>
<tr>
<td>Revision</td>
<td></td>
</tr>
<tr>
<td>SNMP Settings</td>
<td></td>
</tr>
<tr>
<td>Search domains</td>
<td>example.com</td>
</tr>
<tr>
<td>Sign installer certificates</td>
<td>AUTO</td>
</tr>
<tr>
<td>Time servers</td>
<td>0.pool.ntp.org, 1.pool.ntp.org, 2.pool.ntp.org</td>
</tr>
<tr>
<td>Time zone</td>
<td>America/Los_Angeles</td>
</tr>
</tbody>
</table>

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](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**

```bash
[bright92->device[node001]]% history | tail -3
  162 use node001
  163 history
  164 history | tail -3
[bright92->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**

```bash
[root@mycluster ~]# tail -2 .cm/.cmshhistory
1615412046
device list
[root@\bprompt ~]# 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**

```bash
[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**

```bash
[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
```

<table>
<thead>
<tr>
<th>Type</th>
<th>Hostname (key)</th>
<th>MAC (key)</th>
<th>Category</th>
</tr>
</thead>
<tbody>
<tr>
<td>PhysicalNode</td>
<td>node001</td>
<td>00:E0:81:2E:F7:96</td>
<td>default</td>
</tr>
</tbody>
</table>

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
```
[bright92]# device ssh node001
<screen is cleared>
<some MOTD text and login information is displayed>
[root@node001 ~]# exit
Connection to node001 closed.
[bright92]# device use bright92
[bright92->device[bright92]]% # now let us connect to the head node from the head node object
[bright92->device[bright92]]% ssh
<screen is cleared>
<some MOTD text and login information is displayed>
[root@bright92 ~]# exit
logout
Connection to bright92 closed.
[bright92->device[bright92]]%
```

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

[bright92->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

[bright92->device]% watch newnodes
screen clears
Every 2.0s: newnodes Thu Dec 3 13:01:45 2015
No new nodes currently available.

Example

[bright92->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 45) 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:

```bash
[root@bright92 ~]# 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|litename|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 specified 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 44) can also use the following, more elaborate, syntax:

```
<node>,...,<node>,<node>...<node>
```

**Example**

```bash
[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**

```bash
[bright92->device]% foreach --clone node001 -n node022..node024 ()
[bright92->device]% list | grep node
Type Hostname (key) Ip
```
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**

```bash
[bright92->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**

```bash
[myheadnode->device]% foreach * (get ip; status)
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**

```bash
[bright92->category]% foreach * (get name; listnodes)
default
Type Hostname MAC Category Ip Network Status
```

---

<table>
<thead>
<tr>
<th>PhysicalNode</th>
<th>node001</th>
<th>10.141.0.1</th>
</tr>
</thead>
<tbody>
<tr>
<td>PhysicalNode</td>
<td>node022</td>
<td>10.141.0.22</td>
</tr>
<tr>
<td>PhysicalNode</td>
<td>node023</td>
<td>10.141.0.23</td>
</tr>
<tr>
<td>PhysicalNode</td>
<td>node024</td>
<td>10.141.0.24</td>
</tr>
</tbody>
</table>

[bright92->device]% commit
<table>
<thead>
<tr>
<th>Prefix</th>
<th>Type</th>
<th>Description</th>
<th>Example</th>
</tr>
</thead>
<tbody>
<tr>
<td>PhysicalNode</td>
<td>node001</td>
<td>FA:16:3E:79:4B:77 default 10.141.0.1 internalnet [ UP ]</td>
<td></td>
</tr>
<tr>
<td>PhysicalNode</td>
<td>node002</td>
<td>FA:16:3E:41:9E:A8 default 10.141.0.2 internalnet [ UP ]</td>
<td></td>
</tr>
<tr>
<td>PhysicalNode</td>
<td>node003</td>
<td>FA:16:3E:00:1F:E1 default 10.141.0.3 internalnet [ UP ]</td>
<td></td>
</tr>
</tbody>
</table>

The `-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,...,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,r3[3-5]n001-03
    
    which is: r2n003,r3n001,r3n002,...,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@bright92 ~]# cat .cm/cmsh/.cmshrc
```

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**

```
[bright92->device]% range -n node0[01-24]
[bright92->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**

```
[bright92->device{-n node001..024}]% range -c default
[bright92->device{-c default}]% ds
node001 .................. [ UP ] state flapping
node002 .................. [ UP ]
node003 .................. [ UP ]
```

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

```
[bright92->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**

```
[bright92->device{-c default}]% get ip
10.141.0.1
10.141.0.2
10.141.0.3
[bright92->device{-c default}]% ..
[bright92->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**

```
[bright92->device]% range -n node[001-100]
[bright92->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@bright92 ~]# 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.

- 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 19) 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@bright92 ~]# 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 Bright View (section 3.1.1) or in cmsh (section 2.5.4, and also page 78).

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

```
[bright92->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

```
[bright92->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

```
[bright92->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

```
[bright92->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@bright92 ~]# cmsh
[bright92]% device
[bright92->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
[bright92->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@bright92 ~]# cmsh
[bright92]% device
[bright92->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
[bright92->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
[bright92->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 1615466821582
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

[bright92->device]% list -f hostname,ip,mac,status
hostname (key)  ip          mac                      status
-------------------- -------------------- -------------------- -----
node001  10.141.0.1  FA:16:3E:95:80:9F  [ UP ]
bright92 10.141.255.254 FA:16:3E:D3:56:E0  [ UP ]

[bright92->device]% color off; list -f hostname,ip,mac,status -d [
  "hostname (key)": "bright92",
  "ip": "10.141.255.254",
  "mac": "FA:16:3E:D3:56:E0",
  "status": "[ UP ]"
,
  "hostname (key)": "nodes001",
  "ip": "10.141.0.1",
  "mac": "FA:16:3E:95:80:9F",
  "status": "[ UP ]"
]
[bright92->device]%

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

Example

[root@bright92 ~]# 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 Bright 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 17.8).

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:

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

* arguments that work with cmdaemonctl 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@bright92 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
```
Example

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

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

This uses `pdsh`, the parallel shell command (section 17.1).

### 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 Chapter 12.

#### CMDaemon Version Extraction

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

Example
2.6 Cluster Management Daemon

```
[bright92->device]% cmdaemonversions
Hostname  Version index  Version hash
---------  --------------  ----------------
bright92  146,965        e6f59b676
node001   146,965        e6f59b676
node002   146,965        e6f59b676
```

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:

```
[bright92->device]% cmdaemonversions --join
Version index  Version hash  Count  Hostnames
-----------  --------------  -----  ----------------
146,965      e6f59b676      3      bright92,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@bright92 ~]# cmdaemonctl -h
```

Command: `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@bright92 ~]# 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**

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

CMDaemon Subsystem Logging Configuration Deployment

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

**Example**

```
[root@bright92 etc]# service cmd restart
```

or by reloading the logging configuration:

**Example**

```
[root@bright92 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:

```bash
# 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 857) is used to keep it frozen.

Sections of files that have been generated by the cluster management daemon will read as follows:

```bash
# 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**

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

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

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

Using `FrozenFile` can be regarded as a configuration technique (section 3.16.3), and one of various possible configuration techniques (section 3.16.1).
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.3), are given in Appendix A.4.

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

```bash
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@bright92 ~]# scp /cm/shared/apps/cm-lite-daemon-dist/cm-lite-daemon.zip
   root@lite01:/opt/cm-lite-daemon.zip
[root@bright92 ~]# 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
```
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 Bright 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

[bright92->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:

```
bright92->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
```
2.6 Cluster Management Daemon

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

The lite node should then show up as being in the UP state in Bright 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 80) 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
Configuring The Cluster

After the NVIDIA Bright 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 12). 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 InfiniBand and Omni-Path are set up.
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.
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 Bright View—can be used to carry out cluster management (Chapter 2), the Bright 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 Bright View can be used for the global configuration of cluster settings. For the cmsh front end, the configuration is done using partition mode. The analogue to global configuration in the Bright View front end relies on the clickpath: Cluster → Settings.

This section now continues with the Bright View description.
Configuring The Cluster

The clickpath: Cluster → Settings brings up Bright View’s cluster Settings window (figure 3.1):

![Cluster Settings](image)

**Figure 3.1: Cluster Settings**

The navigational overview (figure 2.3) 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 Bright View Settings window, the following defaults can be viewed and modified for names throughout the cluster.

- **Cluster name**: (default name: Bright 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 Bright 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 78) of cmsh.

More specific renaming of node names is possible using the rename command from device mode of cmsh, (section 2.5.5, page 48).

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 Bright 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 Bright View, then the changes to the nameserver and searchdomain values can instead be carried out via partition mode (page 76).

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

[bright92]\% network use ibnet
[bright92->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

[bright92]\% partition
[bright92->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 Bright View, the clickpath is:
  Cluster→Settings→EXTERNALLY VISIBLE IP.
- For `csh`, 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 Bright View, the clickpath is via Cluster→Settings→NAME SERVERS.
- For `csh`, 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 Bright View, the time zone parameters can be jumped to via the clickpath:
  Cluster→Settings→JUMP TO→Time zone
- In `csh`, 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
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 Bright 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: bright)
- **Password**: (default: random string generated during head node installation)
- **User ID**: (default: 4)
- **Power reset delay**: During a reset, this is the time, in seconds, that the machine is off, before it starts powering up again (default: 0)
- **Extra arguments**: (default: none)
- **privilege**: (default: administrator)

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

BMC configuration is discussed in more detail in section 3.7.

**Administrator E-mail Setting**

By default, the distribution which 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 Bright 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:

  ```
  [bright92]% partition use base
  [bright92->partition[base]]% set administrator-mail alf@example.com beth@example.com
  [bright92->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 Bright 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@bright92 ~]# cmsh
[bright92] partition use base
[bright92-> partition[base]]% set relayhost mail.example.com
[bright92-> 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 Advanced Config (page 862)StmtPartitionRelayHostInImages 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@bright92 ~]# 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@bright92 ~]# cmsh -c "main; testemail"
Mail delivered to postfix
You have new mail in /var/spool/mail/root
[root@bright92 ~]# 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 Bright View. This opens up a subwindow that can be used to configure HA-related parameters (section 18.4.6).

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 Bright View. This opens up a subwindow that can be used to configure failover-groups-related parameters (section 18.5).

Burn Configs
Burning nodes is covered in Chapter 8 of the Installation Manual. Burn configuration settings for the cluster can be accessed in Bright 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.
3.1 Main Cluster Configuration Settings

- 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 NVIDIA Bright 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 NVIDIA Bright 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 \texttt{/proc/sys/fs/file-max} Setting
The maximum number of open files allowed on a running Linux operating system is determined by \texttt{/proc/sys/fs/file-max}. To configure this setting so that it is persistent, the Linux operating system uses a \texttt{/etc/sysctl.conf} file and \texttt{*\_conf} files under \texttt{/etc/sysctl.d/}. Further information on these files can be found via the man page, \texttt{man(5) sysctl.conf}. The cluster manager adheres to this standard method, and places a settings file \texttt{90\_cm\_sysctl.conf} in the directory \texttt{/etc/sysctl.d/}.

By default, the value set for \texttt{file-max} by NVIDIA Bright Cluster Manager for versions 8.2 and beyond is 131072. A head node typically is not used to run applications that will exceed this value. However, cluster administrators with larger clusters or with custom needs can change the value according to need.

Configuring The User Limit On Open Files: The \texttt{nofile} Setting
The maximum number of open files allowed for a user can be seen on running \texttt{ulimit -n}. The value is defined by the \texttt{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 \texttt{pam\_limits.so} module is used to apply ulimit limits, then the resource limits can be set via the \texttt{/etc/security/limits.conf} file and \texttt{*\_conf} files in the \texttt{/etc/security/limits.d/} directory. Further information on these files can be found via the man page, \texttt{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 \texttt{nofile} value is defined by the cluster manager in:

- in \texttt{/etc/security/limits.d/91\_cm\_limits.conf} on the head node
- in \texttt{/cm/images/\textless software image name\textgreater /etc/security/limits.d/91\_cm\_limits.conf} in the software image that the regular node picks up.

The values set in \texttt{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 \texttt{nofile} value in \texttt{/etc/security/limits.d/91\_cm\_limits.conf} (but leaving the \texttt{/proc/sys/fs/file-max} untouched), or increasing \texttt{/proc/sys/fs/file-max} (but leaving the \texttt{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 \texttt{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 \texttt{file-max} value. For example, if the node has 128 GB of memory, then 1280000 can be set as the \texttt{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 77 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 73 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.
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 74 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 255.

- 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 Bright View, a network can be configured via the clickpath Networking → Networks, which opens up the Networks subwindow (figure 3.3):

![Networks Table](image)

**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 resource tree displays, for example, by double-clicking the row, opens up the Overview window of that network. 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).
3.2 Network Settings

Selecting the Settings option opens a scrollable pane that allows a number of network properties to be changed (figure 3.5).
The properties of figure 3.5 are introduced in table 3.2.1.

<table>
<thead>
<tr>
<th>Property</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>name</td>
<td>Name of this network.</td>
</tr>
<tr>
<td>Domain Name</td>
<td>DNS domain associated with the network.</td>
</tr>
<tr>
<td>Type</td>
<td>Menu options to set the network type. Options are Internal, External, Tunnel, Global, Cloud, or NetMap network.</td>
</tr>
<tr>
<td>MTU</td>
<td>Maximum Transmission Unit. The maximum size of an IP packet transmitted without fragmenting.</td>
</tr>
<tr>
<td>Node booting</td>
<td>Enabling means nodes are set to boot from this network (useful in the case of nodes on multiple networks). For an internal subnet called &lt;subnet&gt;, when node booting is set, CMDaemon adds a subnet configuration /etc/dhcpd.&lt;subnet&gt;.conf on the head node, which is accessed from /etc/dhcpd.conf.</td>
</tr>
<tr>
<td>Lock down dhcpd</td>
<td>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.</td>
</tr>
<tr>
<td>Management allowed</td>
<td>Enabling means that the network has nodes managed by the head node.</td>
</tr>
<tr>
<td>Base address</td>
<td>Base address of the network (also known as the network address).</td>
</tr>
<tr>
<td>Broadcast address</td>
<td>Broadcast address of the network.</td>
</tr>
<tr>
<td>Dynamic range</td>
<td>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.</td>
</tr>
<tr>
<td>Netmask bits</td>
<td>Prefix-length, or number of bits in netmask. The part after the “/” in CIDR notation.</td>
</tr>
<tr>
<td>Gateway</td>
<td>Default route IP address</td>
</tr>
</tbody>
</table>

Table 3.2.1: 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
3.2 Network Settings

```
user@bright92:~$ 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 Bright 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 Bright 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@bright92 ~]# cmsh
[bright92]# partition use base
[bright92->partition[base]]% set managementnetwork internalnet; commit
[bright92->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 Bright View, the click path 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 Bright 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@bright92 ~]# cmsh
[bright92]% device use bright92
3.2 Network Settings

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

```
[bright92]->device[bright92]% 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@bright92 ~]# sleep 30; hostname -f foobar.cm.cluster
[root@bright92 ~]#
```

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.

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 Bright 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 Bright 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 Bright 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 Bright View or `cmsh`:
With Bright 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:

![Network Settings For External Network](image)

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 bright92, where the IP address is used by the interface eth0, the static IP address can be set via the clickpath Devices → Head Nodes → Edit[bright92] → Settings → JUMP TO Interfaces → Edit[eth0] → IP (figure 3.8).

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

[bright92]% network use externalnet
[bright92->network[externalnet]]% set baseaddress 192.168.1.0
[bright92->network*[externalnet*]]% set netmaskbits 24
[bright92->network*[externalnet*]]% set gateway 192.168.1.1
[bright92->network*[externalnet*]]% commit
[bright92->network[externalnet]]% partition use base
[bright92->partition[base]]% set nameservers 192.168.1.1
[bright92->partition*[base*]]% set searchdomains x.com y.com
[bright92->partition*[base*]]% append timeservers ntp.x.com
[bright92->partition*[base*]]% commit
[bright92->partition[base]]% device use bright92
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 Bright 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:

  \[
  \text{[bright92->device[bright92]->interfaces[eth0]]% set dhcp yes}
  \]

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 Bright 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 77)
2. the cluster network for the internal network (page 78)
3. the individual device network interface (page 79)
4. the node categories network-related items for the device (page 80), 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 Bright 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 Omni-Path 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 Bright 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.
3.2 Network Settings

- 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 80). 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 Bright View, the network for a node, such as a physical node node001, can be configured via a clickpath Devices \( \rightarrow \) Nodes[node001] \( \rightarrow \) 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:

<table>
<thead>
<tr>
<th>Type</th>
<th>Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>Bridge</td>
<td>80</td>
</tr>
<tr>
<td>Bond</td>
<td>70</td>
</tr>
<tr>
<td>Physical</td>
<td>60</td>
</tr>
<tr>
<td>VLAN</td>
<td>50</td>
</tr>
<tr>
<td>Alias</td>
<td>40</td>
</tr>
<tr>
<td>Tunnel</td>
<td>30</td>
</tr>
<tr>
<td>Netmap</td>
<td>20</td>
</tr>
<tr>
<td>BMC</td>
<td>10</td>
</tr>
</tbody>
</table>

- 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 78 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

[bright92] device interfaces node001
[bright92->device[node001]->interfaces]% add physical eth1
[bright92->...->interfaces*[eth1*]]% set network externalnet
[bright92->...->interfaces*[eth1*]]% set additionalhostnames extra01
[bright92->...->interfaces*[eth1*]]% set ip 10.141.1.1
[bright92->...->interfaces*[eth1*]]% commit
[bright92->...->interfaces[eth1]]%
[bright92->...->interfaces]% ..
[bright92->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 Bright 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 Bright 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 Bright
  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 77, 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 G.2.1).

- the device can be made to work with the health check and metric framework. The scripts used in
  the framework will however almost certainly have to be customized to work with the device

After changing network configurations, a reboot of the device is necessary to activate the changes.
Changing The Global Network Parameters For The Cluster

The global network `globalnet` is a unique network used to set up a common name space for all nodes in a cluster in 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 8.6.1 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
```
Configuring The Cluster

```
[root@bright92 ~]# cmsh
[bright92 ]% device use node001
[bright92->device[node001]]%
[bright92->device[node001]]% staticroutes
[bright92->device[node001]->staticroutes]% list

Name (key) Gateway Destination
-------------------------------- ---------------- --------------

[bright92->device[node001]->staticroutes]% add printroute
[bright92->...*[printroute*]]% set gateway 10.141.1.1
[bright92->...*[printroute*]]% set destination 192.168.0.3
[bright92->...*[printroute*]]% set networkdevicename bootif
[bright92->...*[printroute*]]% commit
[bright92->...*[printroute*]]% show
Parameter Value
-------------------------------- -------------------------------
Destination 192.168.0.3/32
Gateway 10.141.1.1
Name printroute
Network Device Name bootif
Revision

[bright92->device[node001]->staticroutes[printroute]]% exit
[bright92->device[node001]->staticroutes]% list

Name (key) Gateway Destination
-------------------------------- ---------------- --------------
printroute 10.141.1.1 192.168.0.3/32

[bright92->device[node001]->staticroutes[printroute]]% exit; exit
[bright92->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 printroute 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@bright92 ~]# 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

--- 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@bright92->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@bright92 ~]# cmsh
[bright92]# device
[bright92->device]# routes
```

<table>
<thead>
<tr>
<th>Node</th>
<th>Destination</th>
<th>Gateway</th>
<th>Netmask</th>
<th>Interface</th>
</tr>
</thead>
<tbody>
<tr>
<td>bright92</td>
<td>0.0.0.0</td>
<td>192.168.200.254</td>
<td>0.0.0.0</td>
<td>eth1</td>
</tr>
<tr>
<td>bright92</td>
<td>10.141.0.0</td>
<td>0.0.0.0</td>
<td>255.255.0.0</td>
<td>eth0</td>
</tr>
<tr>
<td>bright92</td>
<td>169.254.0.0</td>
<td>0.0.0.0</td>
<td>255.255.0.0</td>
<td>eth1</td>
</tr>
<tr>
<td>bright92</td>
<td>192.168.200.0</td>
<td>0.0.0.0</td>
<td>255.255.255.255</td>
<td>eth1</td>
</tr>
<tr>
<td>node001</td>
<td>0.0.0.0</td>
<td>10.141.255.254</td>
<td>0.0.0.0</td>
<td>eth0</td>
</tr>
<tr>
<td>node001</td>
<td>10.141.0.0</td>
<td>0.0.0.0</td>
<td>255.255.0.0</td>
<td>eth0</td>
</tr>
<tr>
<td>node001</td>
<td>169.254.0.0</td>
<td>0.0.0.0</td>
<td>255.255.0.0</td>
<td>eth0</td>
</tr>
</tbody>
</table>

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 42, 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@bright92 ~]# cmsh
[bright92]# device
[bright92->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

[bright92->device]% connections
Node Type Source Port Destination Port State
------- -------- -------- -------- ------------ ----- ----------
bright92 TCP 0.0.0.0 22 0.0.0.0 0 Listening
bright92 TCP 0.0.0.0 25 0.0.0.0 0 Listening
bright92 TCP 0.0.0.0 111 0.0.0.0 0 Listening
bright92 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

[bright92->device]% connections | head -2; connections | grep Listen | grep " 53"
Node Type Source Port Destination Port State
------- -------- ------------------ -------- --------------- -------- -----------
bright92 TCP 10.141.255.254 53 0.0.0.0 0 Listening
bright92 TCP 127.0.0.1 53 0.0.0.0 0 Listening
bright92 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

[bright92->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

[bright92->device]% connectivity
Source Destination Result ID Latency
------- ----------- ------------ ---- ------------
bright92 node001 0k 0 0.4ms
bright92 node002 0k 0 0.3ms
bright92 node003 0k 0 0.4ms
bright92 bright92 0k 0 0.1ms
3.3 Configuring Bridge Interfaces

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

[bright92->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

[bright92->device[node001]->interfaces]% add bridge br0  
[bright92->...->interfaces*[br0*]]% set network internalnet  
[bright92->...->interfaces*[br0*]]% set ip 10.141.1.1  
[bright92->...->interfaces*[br0*]]% show  
Parameter  Value  
Additional Hostnames  no  
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
The **BOOTIF** interface is also a valid interface option.

Currently, the following bridge properties can be set:

- **Spanning Tree Protocol**: 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 Bright 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@bright92 ~]# cmsh
[bright92]% device interfaces node001
[bright92->device[node001]->interfaces]% list
```
3.4 Configuring VLAN interfaces

<table>
<thead>
<tr>
<th>Type</th>
<th>Network device name</th>
<th>IP</th>
<th>Network</th>
</tr>
</thead>
<tbody>
<tr>
<td>physical</td>
<td>BOOTIF [prov]</td>
<td>10.141.0.1</td>
<td>internalnet</td>
</tr>
</tbody>
</table>

Arguments:
- type
  - alias, bmc, bond, bridge, netmap, physical, tunnel, vlan

```
bright92->device[node001]->interfaces] % list -h | tail -4

[bright92->device[node001]->interfaces] % add vlan BOOTIF.1
[bright92->device[node001]->interfaces[BOOTIF.1]] % commit
[bright92->device[node001]->interfaces[BOOTIF.1]] % show
```

Parameter | Value
--- | ---
Additional Hostname | no
DHCP | no
IP | 0.0.0.0
MAC | 00:00:00:00:00:00
Network | BOOTIF.1
Reorder HDR | no
Revision | vlan

```
[bright92->...[BOOTIF.1]] % set network internalnet
[bright92->...[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 Bright View

Within Bright 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).
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

```
[bright92->device[node001]->interfaces] % add bond bond0
[...->device[node001]->interfaces*[bondD*]] % set network internalnet
[...->device[node001]->interfaces*[bondD*]] % set ip 10.141.128.1
[...->device[node001]->interfaces*[bondD*]] % set mode 3
[...->device[node001]->interfaces*[bondD*]] % 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:
3.5 Configuring Bonded Interfaces

- 0 – balance-rr
- 1 – active-backup
- 2 – balance-xor
- 3 – broadcast
- 4 – 802.3ad
- 5 – balance-tlb
- 6 – balance-alb

Technically, outside of Bright 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 18), 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 18.1) 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 18.1) 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 234.

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 Further Notes On Bonding

If using bonding to provide failover services, miimon, which is off by default (set to 0), should be given a value. The miimon setting is the time period in milliseconds between checks of the interface carrier state. A common value is miimon=100.

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
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 NVIDIA Bright 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 installation, then the `rdma` script runs during the `init` stage of booting up for the enabled nodes. For SLES and Linux distributions based on versions prior to Red Hat 6, the `openibd` script is used instead of the `rdma` script.

The `rdma` or `openibd` script takes care of loading the relevant InfiniBand HCA kernel modules. When adding an InfiniBand network after installation, it may be necessary to use `chkconfig` manually to configure the `rdma` or `openibd` script to be run at boot-time on the head node and inside the software images.

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: “`chkconfig opensm on`”. 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.

\(^1\)usually `opensm`, but `openssd` in SLES
In `cmsh` this can be done with:

```
[root@bright92 ~]# cmsh -c "device roles <node>; assign subnetmanager; commit"
```

where `<node>` is the name of a node on which it will run, for example: `bright92, node001, node002`...

In Bright 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 → Nodes → node002 → 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@bright92 ~]# cmsh
[bright92]% device services node001
[bright92->device[node001]->services]% add opensm
[bright92->device[node001]->services[opensm]% set autostart yes
[bright92->device[node001]->services[opensm]% set monitored yes
[bright92->device[node001]->services[opensm]% commit
[bright92->device[node001]->services[opensm]]%
```

In Bright 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 → Nodes → node002 → 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:

<table>
<thead>
<tr>
<th>Property</th>
<th>Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>Name</td>
<td>ibnet</td>
</tr>
<tr>
<td>Domain name</td>
<td>ib.cluster</td>
</tr>
<tr>
<td>Type</td>
<td>internal</td>
</tr>
<tr>
<td>Base address</td>
<td>10.149.0.0</td>
</tr>
<tr>
<td>Netmask bits</td>
<td>16</td>
</tr>
<tr>
<td>MTU</td>
<td>up to 4k in datagram mode</td>
</tr>
<tr>
<td></td>
<td>up to 64k in connected mode</td>
</tr>
</tbody>
</table>

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 `connect-mode` property:

```
Example
```
3.6 Configuring InfiniBand And Omni-Path Interfaces

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@bright92 ~]# 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@bright92 ~]# 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:

[bright92->device[bright92]->interfaces]% add physical ib
[bright92->device[bright92]->interfaces*[ib0%]]% set network ibnet
[bright92->device[bright92]->interfaces*[ib0%]]% set ip 10.149.255.254
[bright92->device[bright92]->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

[bright92]% device add ibswitch myibswitch
[bright92->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
...

[bright92->device[node001]->interfaces[ib0]]% set connectedmode yes
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@bright92 ~]# su - cmsupport
[cmsupport@bright92 ~]# cd /cm/shared/apps/imb/current/
[cmsupport@bright92 current]$/./setup.sh
[cmsupport@bright92 current]$/ cd ~/BenchMarks/imb/2017
[cmsupport@bright92 2017]$/ module load openmpi/gcc
[cmsupport@bright92 2017]$/ module initadd openmpi/gcc
[cmsupport@bright92 2017]$/ make -f make_mpi2
[cmsupport@bright92 2017]$/ 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 21.63 722.25
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 1065.28 996.72
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:

<table>
<thead>
<tr>
<th>Property</th>
<th>Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>Name</td>
<td>bmcnet, ilonet, ipminet, dracnet, cimcnet, or rfnet</td>
</tr>
<tr>
<td>Domain name</td>
<td>bmc.cluster, ilo.cluster, ipmi.cluster, drac.cluster,</td>
</tr>
<tr>
<td></td>
<td>cimc.cluster, or rf.cluster</td>
</tr>
<tr>
<td>External network</td>
<td>false</td>
</tr>
<tr>
<td>Base address</td>
<td>10.148.0.0</td>
</tr>
<tr>
<td>Netmask bits</td>
<td>16</td>
</tr>
<tr>
<td>Broadcast address</td>
<td>10.148.255.255</td>
</tr>
</tbody>
</table>

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:

```
[bright92 ~]# 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:

```
[bright92 ~]# 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@bright92 ~]# cmsh
[bright92]# device
[bright92->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
[bright92->device]% addinterface -n node001..node150 bmc ipmi0 bmcnet 10.148.0.1
[bright92->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:

[bright92->device[bright92]->interfaces]% add alias eth0:0
[bright92->device[bright92]->interfaces*[eth0:0]% set network bmcnet
[bright92->device[bright92]->interfaces*[eth0:0]% set ip 10.148.255.254
[bright92->device[bright92]->interfaces*[eth0:0]% commit
[bright92->device[bright92]->interfaces*[eth0:0]%]
Mon Dec 6 05:45:05 bright92: Reboot required: Interfaces have been modified
[bright92->device[bright92]->interfaces*[eth0:0]% quit
[root@bright92 ~]# 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@bright92 ~]# 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 Configuring BMC (IPMI/iLO/DRAC/CIMC/Redfish) Interfaces

3.7.2 BMC Authentication

The node-installer described in Chapter 5 is responsible for the initialization and configuration of the BMC interface of a device. In addition to a number of network-related settings, the node-installer also configures BMC authentication credentials. By default BMC interfaces are configured with username bright and a random password that is generated during the installation of the head node. The password is stored by CMDaemon. It can be managed from cmsh from within the base object of partition mode, in the bmcsettings submode. This means that by default, each BMC in the cluster has that username and password set during node boot.

For example, the current values of the BMC username and password for the entire cluster can be obtained and changed as follows:

Example

[bright92]% partition use base
[bright92->partition[base]]% bmcsettings
[bright92->partition[base]->bmcsettings]% get username
bright
[bright92->partition[base]->bmcsettings]% get password
Za4ehnilohMa2zew
[bright92->partition[base]->bmcsettings]% set username bmcadmin
[bright92->partition[base]->bmcsettings]% set password
enter new password: ********
retype new password: ********
[bright92->partition[base]->bmcsettings]% commit

In Bright 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:

<table>
<thead>
<tr>
<th>Property</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>BMC User ID</td>
<td>User type. Normally set to 4 for administrator access.</td>
</tr>
<tr>
<td>BMC User Name</td>
<td>User name used when sending a BMC command</td>
</tr>
<tr>
<td>BMC Password</td>
<td>Password for specified user name when sending a BMC command</td>
</tr>
<tr>
<td>BMC Power reset delay</td>
<td>Delay, in seconds, before powering up (default value: 0)</td>
</tr>
<tr>
<td>BMC extra arguments</td>
<td>Extra arguments passed to BMC commands</td>
</tr>
<tr>
<td>BMC privilege</td>
<td>Possible options are</td>
</tr>
<tr>
<td></td>
<td>- administrator</td>
</tr>
<tr>
<td></td>
<td>- callback</td>
</tr>
<tr>
<td></td>
<td>- OEMproprietary</td>
</tr>
<tr>
<td></td>
<td>- operator</td>
</tr>
<tr>
<td></td>
<td>- user</td>
</tr>
</tbody>
</table>

the cluster manager stores the BMC configuration, and uses them:

- 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.

If 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 `racadm` Installation

If Dell was chosen as a hardware option when the cluster manager ISO was created for installation, then the Dell OpenManage utilities are located under `/opt/dell` on the head node.

If Dell is chosen as a hardware option when nodes are configured in the cluster manager installer, 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, which 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@bright92 ~]# cmsh
[bright92]# partition use base
[bright92->partition[base]]% bmcsettings
[bright92->partition[base]->bmcsettings]# get password
12345
[bright92->partition[base]->bmcsettings]# get username
tom
[bright92->partition[base]->bmcsettings]%
```

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

The `bright` user privilege can be raised using the following command:

```
/opt/dell/srvadmin/sbin/racadm -r <DRAC interface IP address> -u root -p <root password> set
iDRAC.Users.4.Privilege 511
```

Here it is assumed that the BMC user has the username `bright`, a userID 4, and the privilege can be set to 511.
3.7.4 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 855) 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 Bright 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@bright92 ~]# cmsh
[bright92] % device
[bright92->device] % set switch01 ip 10.141.253.2
[bright92->device+] % commit
```

3.8.2 Configuring SNMP

### Configuring SNMP Community Strings

Moreover, 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
Configuring The Cluster

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

```
[bright92]% device use switch01
[bright92->device[switch01]]% snmpsettings
[bright92->device[switch01]->snmpsettings]% show
Parameter          Value
----------------------------------
Authentication key       < not set >
Authentication protocol  MD5
Context                  < not set >
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

```
[bright92->device*[switch01*]->snmpsettings*]% set authenticationprotocol <TAB><TAB>
md5 sha
[bright92->device[switch01]->snmpsettings]% set authenticationprotocol aes
[bright92->device*[switch01*]->snmpsettings*]% set privacyprotocol <TAB><TAB>
aes des
[bright92->device*[switch01*]->snmpsettings*]% set privacyprotocol aes
[bright92->device*[switch01*]->snmpsettings*]% commit
[bright92->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@bright92 ~]# cmsh
[bright92]% device use node001
[bright92->device[node001]]% roles
[bright92->device[node001]->roles]% assign snmptrap
[bright92->device[node001]->roles*[snmptrap*]]% show
```

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>Access</td>
<td>public</td>
</tr>
<tr>
<td>Add services</td>
<td>yes</td>
</tr>
<tr>
<td>All administrators</td>
<td>no</td>
</tr>
<tr>
<td>Alternative script</td>
<td></td>
</tr>
<tr>
<td>Arguments</td>
<td></td>
</tr>
<tr>
<td>Event</td>
<td>yes</td>
</tr>
<tr>
<td>Mail</td>
<td>yes</td>
</tr>
<tr>
<td>Name</td>
<td>snmptrap</td>
</tr>
<tr>
<td>Provisioning associations</td>
<td>&lt;0 internally used&gt;</td>
</tr>
<tr>
<td>Recipients</td>
<td></td>
</tr>
<tr>
<td>Revision</td>
<td></td>
</tr>
<tr>
<td>Sender</td>
<td></td>
</tr>
<tr>
<td>Server</td>
<td></td>
</tr>
<tr>
<td>Type</td>
<td>SnmpTrapRole</td>
</tr>
</tbody>
</table>

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, Bright View or `cmsh` are used:

- In Bright View, the switch is selected, and uplinks can be added via the clickpath:

  Devices→Switches→Edit→Uplinks

  (figure 3.11). Clicking on the resulting `Add` button brings up a dialog box that allows port numbers corresponding to uplink port numbers to be filled in. The state is saved with the `Save` button.
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**

```bash
[root@bright92 ~]# cmsh
[bright92] % device
[bright92->device] % set switch01 uplinks 15 16
[bright92->device*] % set switch02 uplinks 01
[bright92->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**
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:

```
[bright92->device]% showport -s switch01 00:30:49:00:73:92
[bright92->device]% 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:

```
#!/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 bright92: 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:

```
[root@bright92 ~]# cmsh
[bright92]% device
[bright92->device]% use node001
[bright92->device[node001]]% clear switchports
[bright92->device[node001]]% commit
```

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

```
[root@bright92 ~]# cmsh
[bright92]% device
[bright92->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):

```
[device]\% switchoverview dell-switch1
Device: dell-switch1
State : [ UP ]
Model : 24G Ethernet Switch

Port Assignment:

<table>
<thead>
<tr>
<th>Port</th>
<th>Status</th>
<th>Assigned</th>
<th>Uplink Speed</th>
<th>Detected</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>UP</td>
<td>false</td>
<td>1 Gb/s</td>
<td></td>
</tr>
<tr>
<td>2</td>
<td>UP</td>
<td>false</td>
<td>1 Gb/s</td>
<td></td>
</tr>
<tr>
<td>3</td>
<td>UP</td>
<td>false</td>
<td>1 Gb/s</td>
<td>74:86:7A:AD:3F:2F, node3</td>
</tr>
<tr>
<td>4</td>
<td>UP</td>
<td>false</td>
<td>1 Gb/s</td>
<td>74:86:7A:AD:43:E9, node4</td>
</tr>
<tr>
<td>...</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>11</td>
<td>UP</td>
<td>false</td>
<td>1 Gb/s</td>
<td>74:86:7A:AD:44:D8, node11</td>
</tr>
<tr>
<td>...</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>50</td>
<td>UP</td>
<td>false</td>
<td>10 Gb/s</td>
<td>74:86:7A:AD:41:A0, node2</td>
</tr>
<tr>
<td>53</td>
<td>UP node34</td>
<td>false</td>
<td>1 Gb/s</td>
<td>5C:F9:DD:F5:79:0D, node34</td>
</tr>
<tr>
<td>54</td>
<td>UP node35</td>
<td>false</td>
<td>1 Gb/s</td>
<td>5C:F9:DD:F5:45:AC, node35</td>
</tr>
<tr>
<td>...</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>180</td>
<td>UP</td>
<td>false</td>
<td>100 Mb/s</td>
<td>QDR-switch</td>
</tr>
<tr>
<td>205</td>
<td>UP</td>
<td>true</td>
<td>10 Gb/s</td>
<td></td>
</tr>
<tr>
<td>206</td>
<td>DOWN</td>
<td>false</td>
<td>10 Gb/s</td>
<td></td>
</tr>
</tbody>
</table>
```

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)
3.9 Disk Layouts: Disked, Semi-Diskless, And Diskless Node Configuration

- 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 Bright 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 Bright View or `cmsh` as follows:

**Changing A Disk Layout Using Bright View**

To change a disk layout with Bright View, the current disk layout is accessed by selecting a node category or a specific node from the resource tree. In the `Settings` pane, the `Disk setup` field can be edited. Clicking on the `Browse` button shows several possible configurations that can be loaded up, and if desired, edited to suit the situation (figure 3.12). To switch from the existing disk layout to a diskless one,

![Figure 3.12: Changing A Disked Node To A Diskless Node With Bright View](image)

the diskless XML configuration is loaded and saved to the node or node category.
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@bright92 ~]# cmsh
[bright92] % device use node001
[bright92->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

[bright92->device[node001]] % set disksetup /cm/local/apps/cmd/etc/htdocs/disk-setup/slave-diskless.xml

In these preceding Bright 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.

In RHEL7 only the client kernel module is provided for NFS over RDMA by default. The module is built with the compile option:

CONFIG_SUNRPC_XPRT_RDMA_CLIENT
The server module is not provided by default, and must be compiled by the administrator explicitly for the NFS server node. The module can be built by using the compile option:

```
CONFIG_SUNRPC_XPRT_RDMA_SERVER
```

However, this is currently (January 2015) unstable and unsupported by Red Hat. The server module may be useable for custom-built kernel versions beyond 3.17. The Bright Computing suggestion for an administrator who needs a very fast network filesystem 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 Bright 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@bright92 ~]# cmsh
[bright92]# device use bright92
[bright92->device[bright92]]# services
[bright92->device[bright92]->services[]]# add autofs
[bright92->device[bright92]->services*[autofs*]]# show

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>Autostart</td>
<td>no</td>
</tr>
<tr>
<td>Belongs to role</td>
<td>no</td>
</tr>
<tr>
<td>Monitored</td>
<td>no</td>
</tr>
<tr>
<td>Run if</td>
<td>ALWAYS</td>
</tr>
<tr>
<td>Service</td>
<td>autofs</td>
</tr>
<tr>
<td>Sickness check interval</td>
<td>60</td>
</tr>
<tr>
<td>Sickness check script</td>
<td>autofs</td>
</tr>
<tr>
<td>Sickness check script timeout</td>
<td>10</td>
</tr>
<tr>
<td>Timeout</td>
<td>-1</td>
</tr>
</tbody>
</table>
```

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 247.

The rest of this section describes the configuration of NFS for static mounts, using Bright 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 Bright 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 Bright View And cmsh

Exporting A Filesystem Using Bright View

As an example, if an NFS volume exists at “bright92:/modeldata” it can be exported using Bright View as follows:

The bright92 head node is selected via the clickpath Devices→Head nodes[bright92]→Settings→Filesystem exports. This shows the list of exports (figure 3.13):

![Figure 3.13: NFS Exports From A Head Node Viewed Using Bright View](image)

Using the Add button, a new entry (figure 3.14) can be configured with values as shown:
3.10 Configuring NFS Volume Exports And Mounts

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

**Example**

```
[root@bright92 ~]# cmsh
[bright92]% device use bright92
[bright92->device[bright92]]% fsexports
[...->fsexports]% add "Fluid Model Data"
[...->fsexports[Fluid Model Data]]% set path /modeldata
[...[Fluid Model Data]]% set hosts 10.141.0.0/16
[...[Fluid Model Data]]% commit
[...->fsexports[Fluid Model Data]]% list | grep Fluid
            Name (key)  Path         Hosts          Write
-----------------------------------------------            -------
Fluid Model Data /modeldata  10.141.0.0/16      no
```

**General Considerations On Exporting A Filesystem**

**Built-in exports:** In versions of NVIDIA Bright 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 NVIDIA Bright 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@bright92 ~]# cmsh
[bright92]% device use bright92
[bright92->device[bright92]]% roles
[bright92->device[bright92]->roles]% use headnode
[bright92->device[bright92]->roles[headnode]]% show
Parameter            Value
------------------- ---------------
Name                 headnode
Type                 HeadNodeRole
Add services         yes
Disable automatic exports no
Provisioning associations <2 internally used>
role use headnode
[bright92->device[bright92]->roles[headnode]]% set disableautomaticexports yes ; commit
```
Disabling exports that are not built-ins: Exports that are not built-ins can still simply be removed. However, also from NVIDIA Bright Cluster Manager 9.0 onwards, they can also simply be disabled in the \texttt{fsexports} submode. For example there is an export created by the cluster administrator for /opt, it can be disabled as follows:

\textbf{Example}

\begin{verbatim}
[bright92->device[bright92]->fsexports]% list
Name (key) Path Network Disabled
---------------- ------- ... ----------- --------
opt /opt internalnet no
[bright92->device[bright92]->fsexports]% set opt disabled yes; commit
\end{verbatim}

The reason for automating the export for nodes via a headnode role is that NVIDIA Bright Cluster Manager 9.0 onward has multidistro and multiarch capabilities (section 12.7), which would make manual management of exports harder for such nodes. The reason for the extra hurdle of \texttt{disableautomaticexports} for built-ins is that that disabling these exports can result in an unbootable system.

\subsection*{3.10.2 Mounting A Filesystem Using Bright View And \texttt{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 \texttt{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.

\textbf{Mounting A Filesystem Using Bright View}
Thus, in Bright View, values for the remote mount point (bright92:/modeldata), the filesystem type (nfs), and the local mount point (/modeldatagpu) can be set in category mode, while the remaining options stay at their default values (figure 3.15):
Saving the configuration saves the values, creating the local mount point, and the volume can now be accessed by nodes in that category.

**Mounting A Filesystem Using cmsh**

The equivalent to the preceding Bright 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

[bright92->category[default]]% category use default
[bright92->category[default]]% fsmounts
[bright92->category[default]->fsmounts]% add /modeldatagpu
[bright92->category[default]->fsmounts]% set device bright92:/modeldata
[bright92->category[default]->fsmounts]% set filesystem nfs
[bright92->category[default]->fsmounts]% commit
```

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>Device</td>
<td>/modeldata</td>
</tr>
<tr>
<td>Mountpoint (key)</td>
<td>/modeldatagpu</td>
</tr>
<tr>
<td>Filesystem</td>
<td>nfs</td>
</tr>
</tbody>
</table>

...
### 3.10 Configuring NFS Volume Exports And Mounts

Device: bright92:/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:

```
[bright92->...[/modeldatagpu]]% set mountoptions defaults,noac; commit
```

Mounting a CIFS might use:

```
[bright92->...[/modeldatagpu]]% set mountoptions gid,users,file_mode=0666,dir_mode=0777,iowcharset=iso8859-15,credentials=/path/to/credential
[bright92->...[/modeldatagpu*]]% commit
```

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

```
[bright92->...[/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 17.1), 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 Bright 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@bright92 ~]# cmsh -c "device; fsmounts node001; list"
```

<table>
<thead>
<tr>
<th>Device</th>
<th>Mountpoint (key)</th>
<th>Filesystem</th>
</tr>
</thead>
<tbody>
<tr>
<td>[default] none</td>
<td>/dev/pts</td>
<td>devpts</td>
</tr>
<tr>
<td>[default] none</td>
<td>/proc</td>
<td>proc</td>
</tr>
<tr>
<td>[default] none</td>
<td>/sys</td>
<td>sysfs</td>
</tr>
<tr>
<td>[default] none</td>
<td>/dev/shm</td>
<td>tmpfs</td>
</tr>
<tr>
<td>[default] $localnfsserv+</td>
<td>/cm/shared</td>
<td>nfs</td>
</tr>
<tr>
<td>[default] bright92:/home</td>
<td>/home</td>
<td>nfs</td>
</tr>
<tr>
<td>bright92:/cm/shared/exa+</td>
<td>/home/examples</td>
<td>nfs</td>
</tr>
</tbody>
</table>

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> on the provisioning node, and the subtree is mounted accordingly under / on the diskless node.

While there are no restrictions placed on the paths that may be mounted in NVIDIA Bright Cluster Manager 9.2, the administrator should be aware that mounting certain paths such as /bin is not possible.

When Bright 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@bright92 ~]# cmsh
[bright92]% device use bright92
[bright92->device[bright92]->fsexports]% add defaultimage
[bright92...defaultimage*]% set path /cm/images/default-image
[bright92...defaultimage*]% set hosts 10.141.0.0/16
[bright92...defaultimage*]% commit
[bright92...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):

```
[bright92...defaultimage%]% category use default
[bright92->category[default]->fsmounts]% add /usr/share/doc
[bright92->...[/usr/share/doc%]% set device bright92:/cm/images/default-image/user/share/doc
[bright92->...[/usr/share/doc%]% set filesystem nfs
[bright92->category[default]->fsmounts[/usr/share/doc%]% commit
[bright92->category[default]->fsmounts[/usr/share/doc%]% list
Device                Mountpoint (key)   Filesystem
---------------------- ------------------ ----------
...     ...     ...
bright92:/cm/images/usr/share/doc /usr/share/doc   nfs
```

```
[bright92->category[default]->fsmounts[/usr/share/doc%]% show
Parameter        Value
---------------- -----------------------------------------------
Device           bright92:/cm/images/default-image/usr/share/doc
Dump             no
Filesystem       nfs
Filesystem Check 0
Mount options    defaults
Mountpoint       /usr/share/doc
```

\(^{2}\)by default <image> is default-image on a newly-installed cluster
Other mount points can also be added according to the judgment of the system administrator. Some consideration of mount order may be needed, as discussed on page 114 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@bright92 ~]# cd /cm/images/default-image/
[root@bright92 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@bright92 default-image]# cmsh
[bright92]% category use default; fsmounts
[bright92->category[default]->fsmounts]% list -f device:53,mountpoint:17
device             mountpoint (key)
------------------ ------------------------
... ...
master:/cm/shared  /cm/shared
master:/home       /home
bright92:/cm/images/default-image/usr/share/locale /usr/share/locale
bright92:/cm/images/default-image/usr/lib/jvm /usr/lib/jvm
bright92:/cm/images/default-image/usr/share/doc /usr/share/doc
bright92:/cm/images/default-image/usr/src /usr/src
[bright92->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@bright92 ~]# cmsh
[bright92]% category use default
[bright92->category[default]]% set disksetup /cm/images/default-image/cm/local/apps/cmd/etc/htdocs/disk-setup/slave-diskless.xml
[bright92->category[default]]% commit  #puts full OS in RAM
```

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

```
[bright92->category[default]]% device use master; fsexports
[bright92->device[bright92]->fsexports]% add /cm/images/default-image ...
```
3.10 Configuring NFS Volume Exports And Mounts

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

```
[root@bright92 ~]# 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 3.16.4) 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@bright92 ~]# 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 UGE/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@bright92 ~]# (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
Configuring The Cluster

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 18) 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 (ibrdcm), 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/nfs.mount.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 Bright Computing 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 Bright 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 Bright View is much the same as done in section 3.10.1 ("Exporting A Filesystem Using Bright 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 Bright View And cmsh Using NFS Over RDMA

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

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

```
chkconfig
```

```
and
```

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

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

Already installed services can also be brought under cluster manager control, and started and stopped with Bright 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 73.

- `runif`: (only honored for nodes used as part of a high availability configuration (chapter 18)) 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 18.5.3).

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. An init script sample can be found within the file /usr/share/doc/initscripts-<version>/sysvinitfiles in RHEL7 and derivatives.

• 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 will still start up on reboot. Running chkconfig 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 18.4.6). 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@bright92 ~]# chkconfig cups on
... symlinks created...
[root@bright92 ~]# 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):

```
[bright92]% device services node001
[bright92->device[node001]->services]% add cups
[bright92->device*[node001*]->services*[cups*]]% show
Parameter       Value
----------------- ------------------------
Autostart        no
Belongs to role  no
Monitored        no
...
Run if           ALWAYS
Service          cups
...
[bright92->device*[node001*]->services*[cups*]]% set monitored on
[bright92->device*[node001*]->services*[cups*]]% set autostart on
[bright92->device*[node001*]->services*[cups*]]% commit
[bright92->device[node001]->services[cups]]
Apr 14 14:02:16 2017 [notice] node001: Service cups was started
[bright92->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:

```
[bright92->device[node001]->services[cups]]% stop
Fri Apr 14 14:03:40 2017 [notice] node001: Service cups was stopped
```

Successfully stopped service cups on: node001

```
[bright92->device[node001]->services[cups]]% status

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

```
[bright92->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 Bright View” on page 123.

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

```
[bright92->device[node001]->services[cups]]% category
[bright92->category]% services default
[bright92->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**

```
[bright92->category*[default*]->services*[cups*]]% show
...
[bright92->category*[default*]->services*[cups*]]% set autostart yes
[bright92->category*[default*]->services*[cups*]]% set monitored yes
[bright92->category*[default*]->services*[cups*]]% commit
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

[bright92->category*[default*]->services*[cups*]]% status

  node001................. cups [STOPPED ]
  node002................. cups [ UP   ]
  node003................. cups [ UP   ]
  node004................. cups [ UP   ]
  node005................. cups [ UP   ]
```

**Managing The Service From Bright View**

Using Bright View, a service can be managed from a Services option, accessible from the Settings option of

- Head Nodes, for example via a clickpath of
  Devices→Head Nodes[bright92]→Settings→Services
- Nodes, for example via a clickpath of
  Devices→Nodes[node001]→Settings→Services
• **Node categories**, for example via a clickpath of
  
  *Grouping* → *Node categories*[default] → *Settings* → *Services*

Figure 3.16 shows the Services subwindow accessed from the default software image, which is an item within the “Node Categories” subwindow. By default, there are no services set by the category for nodes.

Figure 3.16: Operating System Service Configurations Subwindow In Bright View

The *service* `<service name>` `start|stop|restart...` command options `start`, `stop`, `restart`, and so on, are displayed as selection options to the `service` action option.

The existence of the service itself can be managed using the Add, Remove, and, if the service already exists, the Edit button. The change can be reverted with the Revert button.

Figure 3.17 shows CUPS being set up from an Add dialog in the services subwindow. The subwindow is accessible via the `+ ADD` button of figure 3.16.
For a service in the services subwindow, clicking on the Status button in figure 3.16 displays a grid of the state of services on a running node as either Up or Down.

**Monitoring A Service With cmsh And Bright 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 Bright View) to clear a **FAILING** state of a service, reset the attempted auto-starts count to zero, and attempt a restart of the service.

The monitoring system sets the ManagedServicesOk health check (Appendix G.2.1) to a state of **FAIL** if any of the services it monitors is in the **FAILING** state. In cmsh, the statuses of the services are listed by running the `latesthealthdata` command (section 13.6.3) 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:

```bash
[bright92->category[default]->services]% add cups
[bright92->category[default]->services[cups]]% set monitored on
[bright92->category[default]->services[cups]]% set autostart on
[bright92->category[default]->services[cups]]% commit; exit
```

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.

```bash
[bright92->category[default]->services]% add cups
[bright92->category[default]->services[cups]]% set monitored on
[bright92->category[default]->services[cups]]% set autostart off
[bright92->category[default]->services[cups]]% commit; exit
```

Wed May 23 12:54:40 2012 [notice] node001: Service cups was started

```bash
[bright92->category[default]->services]% remove cups; commit
[bright92->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.

Racks Overview

Racks overview in Bright View: The Racks overview pane can be opened up in Bright View via the clickpath Datacenter Infrastructure→Racks (figure 3.18). Racks can then be added, removed, or edited from that pane.

Within the Racks pane:

- a new rack item can be added with the + ADD button.
- an existing rack item can be edited with the Edit button, or by double-clicking on the item itself in the pane.
These actions bring up the rack configuration pane (figure 3.19).

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

```
[bright92->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 Bright View:** The Settings pane for editing a rack item selected from the Racks pane is shown in figure 3.19.

![Figure 3.19: Rack Configuration Settings Using Bright View](image)

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.

• **Bottom of rack is position 1**: 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 use 1 to mark the bottom instead. Ticking the checkbox records the numbering layout accordingly for all racks, if the checkboxed 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**

```
[bright92->rack]# set rack
rack1 rack2 rack3
[bright92->rack]# set rack2

<table>
<thead>
<tr>
<th>height</th>
<th>name</th>
<th>revision</th>
<th>width</th>
<th>y-coordinate</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>inverted</td>
<td>notes</td>
<td>room</td>
<td>x-coordinate</td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
```

The configuration settings for a particular rack obviously match with the parameters associated with and discussed in figure 3.19. The only slightly unobvious match is the Boolean parameter `inverted` in cmsh, which simply corresponds directly to “**Bottom of rack is position 1**” in Bright View.

Setting the values can be done as in this example:

**Example**

```
[bright92->rack]# use rack2
[bright92->rack[rack2]]# set room "skonk works"
[bright92->rack[rack2]]# set x-coordinate 2
[bright92->rack[rack2]]# set y-coordinate 0
[bright92->rack[rack2]]# set inverted no
[bright92->rack[rack2]]# commit
```

### 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 Bright View

Using Bright 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 3.20):
Assigning Devices To A Rack Using cmsh

Using cmsh, the assignment can be done to a rack as follows:

```
[bright92->device]% foreach -n node001..node003 (set deviceheight 1; set deviceposition 2; \ set rack rack2)
[bright92->device*]% commit
Successfully committed 3 Devices
[bright92->device]%
```

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 Bright 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

Basic chassis configuration and node assignment with Bright View: Adding and removing chassis from a list of chassis in Bright View can be done via the clickpath Devices → Chassis. This opens up a Chassis pane that provides an overview of the list of chassis.

Within the Chassis pane, each chassis can further have an individual subpane opened for it. Each individual chassis configuration can be added, edited, or deleted via this chassis subpane (figure 3.21).

![Figure 3.21: Bright View Chassis Subpane](image_url)

The options that can be set within the subpane include the following:

- **Hostname**: a name that can be assigned to the chassis operating system
- **Mac**: the MAC address of the chassis
- **Rack**: the rack in which the chassis is placed
- **Members**: the Members menu option allows devices to be assigned to a chassis (figure 3.22). 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 3.22: Bright View Chassis Members Menu Options](image)

**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

```
[bright92->device[chassis1]]% set
containerindex hostname partition tag
custompingscript ip password userdefined1
custompingscriptargument layout powercontrol userdefined2
custompowerscript mac powerdistributionunits userdefinedresources
custompowerscriptargument members rack username
defaultgateway model revision
deviceheight network slots
deviceposition notes switchports
```

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

```
[bright92->device[chassis1]]% set rack rack2
[bright92->device[chassis1]]% set deviceposition 1; set deviceheight 6
[bright92->device[chassis1]]% commit
```

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

Example

```
[bright92->device[chassis1]]% append members bright92 node001..node005
[bright92->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, but there is an example configuration given at https://community.brightcomputing.com/question/5e16000ef396761939c7ee42/answer for the Dell PowerEdge C410x.

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@bright92 ~]# cmsh
[bright92]% device use node001
[bright92->device[node001]]% gpusettings
[bright92->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
  - *

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@bright92 ~]# cmsh
[bright92]% category use default
[bright92->category[default]]% gpusettings
[bright92->category[default]->gpusettings]% list
GPU range (key) Power limit ECC mode Compute mode Clock speeds
---------------- ------------ ------------ ------------- -------------
[bright92->category[default]->gpusettings]% add nvidia 1-3 ; commit
```
### 3.13 Configuring GPU Settings

```bash
[bright92->category[default]->gpusettings[nvidia:1-3]]% show
```

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>Clock speeds</td>
<td></td>
</tr>
<tr>
<td>Clock sync boost mode</td>
<td></td>
</tr>
<tr>
<td>Compute mode</td>
<td></td>
</tr>
<tr>
<td>ECC mode</td>
<td></td>
</tr>
</tbody>
</table>

...  

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 927), and corrected automatically.
  - Double bit errors are also detected, using the `EccDBitGPU` metric (page 927), 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

```
[bright92->device[node001]->gpusettings]% list
Type   GPU range Info
------ --------- ------------------
AMD 0   PowerPlay: manual

[bright92->device[node001]->gpusettings]% use amd:0
[bright92->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
3.13 Configuring GPU Settings

- **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**

```
[bright92->device[node001]->gpusettings[amd:0]]% status
```

<table>
<thead>
<tr>
<th>Index</th>
<th>Name</th>
<th>Property</th>
<th>Value</th>
<th>Supported</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>Radeon Instinct MI25 Clock</td>
<td>1399Mhz</td>
<td>852Mhz, 991Mhz, ..., 1440Mhz, 1515Mhz</td>
<td></td>
</tr>
<tr>
<td>0</td>
<td>Radeon Instinct MI25 Memory</td>
<td>945Mhz</td>
<td>167Mhz, 500Mhz, 800Mhz, 945Mhz</td>
<td></td>
</tr>
</tbody>
</table>

**The gpusettings Submode In Bright View**

In Bright View the GPU settings can be accessed within the settings options for a device with a GPU. For example, if a regular node, node001, has a GPU, then the clickpath would be:

Devices → Nodes[node001] → Edit → Settings → GPU Settings

which opens up the GPU Settings subpane for that node (figure 3.23). Similarly, GPU settings can also be accessed within the Category resource, selecting a category item, and then selecting the GPU Settings subpane.

**GPU Configuration For HPC Workload Managers**

**Slurm GPU configuration via direct slurm.conf changes**: To configure NVIDIA GPUs for Slurm, changes are made in `slurm.conf` when `cm-wlm-setup` configures GPUs for Slurm (section 7.3).

Changes made are kept in the `AUTOGENERATED` section and can be worked out by checking the difference between the `slurm.conf.template` file and the actual `slurm.conf` file. Changes made include defining the GresTypes `gpu` and `mps`, and setting GPU plugins that allow Slurm generic resources to work.

The configured gres options can be seen by running `sbatch --gres=help`:

**Example**
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:

```bash
#SBATCH --gres=gpu:1
```

Similarly, MPS resources (https://slurm.schedmd.com/gres.html#MPS_Management) can be requested with:

```bash
#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**

```bash
[bright92->configurationoverlay]% list
Name (key)     Priority  All head nodes Nodes Categories  Roles
---------------------------------------------------------------
slurm-accounting 500         yes       slurmaccounting
slurm-client     500         no        slurmclient
slurm-server     500         yes       slurmserver
slurm-submit     500         no        slurmsubmit
wlm-headnode-submit 600       yes       slurmsubmit
```

```bash
[bright92->configurationoverlay]% use slurm-client
[bright92->configurationoverlay[slurm-client]]% roles
```
3.13 Configuring GPU Settings

```
[slurm-client]->roles% use slurmclient
[slurm-client]->roles% genericresources
[slurm-client]->roles->genericresources% add gpu0
[slurm-client]->roles->genericresources[gpu0%] set name gpu
[slurm-client]->roles->genericresources[gpu0%] set file /dev/nvidia0
[slurm-client]->roles->genericresources[gpu0%] commit

(Repeat similar settings for the other GPUs, gpu1...gpu7)
```

```
[slurm-client]->roles% list

<table>
<thead>
<tr>
<th>Alias (key)</th>
<th>Name</th>
<th>Type</th>
<th>Count</th>
<th>File</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>gpu0</td>
<td>gpu</td>
<td></td>
<td>/dev/nvidia0</td>
</tr>
<tr>
<td></td>
<td>gpu1</td>
<td>gpu</td>
<td></td>
<td>/dev/nvidia1</td>
</tr>
<tr>
<td></td>
<td>gpu2</td>
<td>gpu</td>
<td></td>
<td>/dev/nvidia2</td>
</tr>
<tr>
<td></td>
<td>gpu3</td>
<td>gpu</td>
<td></td>
<td>/dev/nvidia3</td>
</tr>
<tr>
<td></td>
<td>gpu4</td>
<td>gpu</td>
<td></td>
<td>/dev/nvidia4</td>
</tr>
<tr>
<td></td>
<td>gpu5</td>
<td>gpu</td>
<td></td>
<td>/dev/nvidia5</td>
</tr>
<tr>
<td></td>
<td>gpu6</td>
<td>gpu</td>
<td></td>
<td>/dev/nvidia6</td>
</tr>
<tr>
<td></td>
<td>gpu7</td>
<td>gpu</td>
<td></td>
<td>/dev/nvidia7</td>
</tr>
</tbody>
</table>
```

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**

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

**Slurm GPU configuration via auto-detection:** Instead of carrying out Slurm configuration by modifying `slurm.conf` by hand, it may be configured via auto-detection. More details on this are to be found starting at page 319.

**PBS:** NVIDIA Bright Cluster Manager version 9.0 onwards supports GPU configuration in PBS via the `cm-wlm-setup` tool after installation (section 7.3.2).

**UGE:** GPU configuration is carried out as follows:

1. **UGE cgroups parameter modifications for GPU devices (obsolete)**

   In NVIDIA Bright 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 \n   mount=false freezer=true freeze_pe_tasks=true \n   killing=false forced_numa=false \n   h_vmem_limit=false m_mem_free_hard=false \n   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 862) to include the setting `UGEDisableCgroupSettings=1` and restarting CMDaemon.

2. GPU devices configuration in UGEClient role (current)

In NVIDIA Bright Cluster Manager 9.0: setting the AdvancedConfig directive (page 862) to include the setting `UGEDisableCgroupSettings=1` is not needed. This is because the `qconf -mconf` configuration method is not required.

Instead, NVIDIA Bright Cluster Manager 9.0 introduces specification of the GPU devices via the `devices` parameter for the UGEClient role:

Example

```
[bright92->category[dgx]->roles[ugeclient]]% set gpus 8
[bright92->category[dgx]->roles[ugeclient]]% set gpudevices gpu0[device=/dev/nvidia0,cuda_id=0]
[bright92->category[dgx]->roles[ugeclient]]% append gpudevices gpu1[device=/dev/nvidia1,cuda_id=1]
[bright92->category[dgx]->roles[ugeclient]]% append gpudevices gpu2[device=/dev/nvidia2,cuda_id=2]
[bright92->category[dgx]->roles[ugeclient]]% append gpudevices gpu3[device=/dev/nvidia3,cuda_id=3]
[bright92->category[dgx]->roles[ugeclient]]% append gpudevices gpu4[device=/dev/nvidia4,cuda_id=4]
[bright92->category[dgx]->roles[ugeclient]]% append gpudevices gpu5[device=/dev/nvidia5,cuda_id=5]
[bright92->category[dgx]->roles[ugeclient]]% append gpudevices gpu6[device=/dev/nvidia6,cuda_id=6]
[bright92->category[dgx]->roles[ugeclient]]% append gpudevices gpu7[device=/dev/nvidia7,cuda_id=7]
[bright92->category[dgx]->roles[ugeclient]]% commit
```

3. UGE port configuration for DCGM

The DCGM port should be set correctly.

In NVIDIA Bright 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.

In NVIDIA Bright Cluster Manager 9.0 onwards, setting the DCGM port for UGE 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

```
[bright92->wlm[lsf]]% set gpuautoconfig yes
[bright92->wlm[lsf]]%
```
The parameter can also be set during LSF configuration via `cm-wlm-setup` (figure 3.24):

![Figure 3.24: GPU settings screen for LSF in cm-wlm-setup](image)

3.13.3 MIG Configuration

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. Once enabled, the full physical GPU is no longer available as a device.

As a sanity check, 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@bright92:~# ssh node001 "nvidia-smi" | grep MIG
| | MIG M. |
| MIG devices: |
| GPU GI CI MIG Memory-Usage Vol Shared |
```

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 1/8th 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 1/7th 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**

```
[bright92->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**

```
[bright92->device[node001]]% mig enable

Node  GPU  Active  Pending
-------- -------- ------------------------
node001  0    no              yes
node001  1    no              yes
node001  2    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):

```
[bright92->device[node001]]% mig disable 3,5-7
```

<table>
<thead>
<tr>
<th>Node</th>
<th>GPU</th>
<th>Active</th>
<th>Pending</th>
</tr>
</thead>
<tbody>
<tr>
<td>node001</td>
<td>0</td>
<td>no</td>
<td>yes</td>
</tr>
<tr>
<td>node001</td>
<td>1</td>
<td>no</td>
<td>yes</td>
</tr>
<tr>
<td>node001</td>
<td>2</td>
<td>no</td>
<td>yes</td>
</tr>
<tr>
<td>node001</td>
<td>3</td>
<td>no</td>
<td>no</td>
</tr>
<tr>
<td>node001</td>
<td>4</td>
<td>no</td>
<td>yes</td>
</tr>
<tr>
<td>node001</td>
<td>5</td>
<td>no</td>
<td>no</td>
</tr>
<tr>
<td>node001</td>
<td>6</td>
<td>no</td>
<td>no</td>
</tr>
<tr>
<td>node001</td>
<td>7</td>
<td>no</td>
<td>no</td>
</tr>
</tbody>
</table>

**Rebooting The MIG Instances To Activate/Deactivate Instances According To Pending State Settings**

The pending states only become active after the node is rebooted:

**Example**

```
[bright92->device[node001]]% reboot
Reboot in progress for: node001
...
node001 [ UP ]
[bright92->device[node001]]% mig status
```

<table>
<thead>
<tr>
<th>Node</th>
<th>GPU</th>
<th>Active</th>
<th>Pending</th>
</tr>
</thead>
<tbody>
<tr>
<td>node001</td>
<td>0</td>
<td>yes</td>
<td>yes</td>
</tr>
<tr>
<td>node001</td>
<td>1</td>
<td>yes</td>
<td>yes</td>
</tr>
<tr>
<td>node001</td>
<td>2</td>
<td>yes</td>
<td>yes</td>
</tr>
<tr>
<td>node001</td>
<td>3</td>
<td>no</td>
<td>no</td>
</tr>
<tr>
<td>node001</td>
<td>4</td>
<td>yes</td>
<td>yes</td>
</tr>
<tr>
<td>node001</td>
<td>5</td>
<td>no</td>
<td>no</td>
</tr>
<tr>
<td>node001</td>
<td>6</td>
<td>no</td>
<td>no</td>
</tr>
<tr>
<td>node001</td>
<td>7</td>
<td>no</td>
<td>no</td>
</tr>
</tbody>
</table>

The effect of `enable` and `disable` persists after reboots, until the pending value changes once more.

**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**

```
[bright92->device[node001]]% mig profiles
```

<table>
<thead>
<tr>
<th>Node</th>
<th>GPU</th>
<th>ID</th>
<th>Name</th>
<th>Instances</th>
<th>Memory</th>
</tr>
</thead>
<tbody>
<tr>
<td>node001</td>
<td>0</td>
<td>1</td>
<td>1g.5gb</td>
<td>7</td>
<td>4.7GiB</td>
</tr>
<tr>
<td>node001</td>
<td>0</td>
<td>1</td>
<td>1g.5gb+me</td>
<td>1</td>
<td>4.7GiB</td>
</tr>
</tbody>
</table>
A list for physical GPU 1 can be specified with:

Example

```
[bright92->device[node001]]% mig profiles 1

<table>
<thead>
<tr>
<th>Node</th>
<th>GPU ID</th>
<th>Name</th>
<th>Instances</th>
<th>Memory</th>
</tr>
</thead>
<tbody>
<tr>
<td>node001</td>
<td>1</td>
<td>1g.5gb</td>
<td>7</td>
<td>4.7GiB</td>
</tr>
<tr>
<td>node001</td>
<td>1</td>
<td>1g.5gb+me</td>
<td>1</td>
<td>4.7GiB</td>
</tr>
<tr>
<td>node001</td>
<td>1</td>
<td>2g.10gb</td>
<td>3</td>
<td>9.7GiB</td>
</tr>
<tr>
<td>node001</td>
<td>1</td>
<td>3g.20gb</td>
<td>1</td>
<td>19.6GiB</td>
</tr>
<tr>
<td>node001</td>
<td>1</td>
<td>4g.20gb</td>
<td>1</td>
<td>19.6GiB</td>
</tr>
<tr>
<td>node001</td>
<td>1</td>
<td>7g.40gb</td>
<td>1</td>
<td>39GiB</td>
</tr>
</tbody>
</table>
```

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

```
[bright92->device[node001]]% gpusettings
[bright92->device[node001]->gpusettings]% list

<table>
<thead>
<tr>
<th>Type</th>
<th>GPU range</th>
<th>Info</th>
</tr>
</thead>
<tbody>
<tr>
<td>Nvidia 0</td>
<td>default</td>
<td></td>
</tr>
</tbody>
</table>
```

```
[bright92->device[node001]->gpusettings]% use nvidia:0
[bright92->device[node001]->gpusettings[nvidia:0]]% show
```

```
<table>
<thead>
<tr>
<th>Parameter</th>
<th>Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>MIG profiles</td>
<td></td>
</tr>
</tbody>
</table>
```

...
• 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

[bright92->device[node001]->gpusettings[nvidia:0]]% set migprofiles 1g.5gb; commit
[bright92->device[node001]->gpusettings[nvidia:0]]% show

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>...</td>
<td></td>
</tr>
<tr>
<td>MIG profiles</td>
<td>1g.5gb</td>
</tr>
</tbody>
</table>

Setting the profiles, and carrying out the commit configures the GPU instances, but does not yet apply them:

Example

[bright92->device[node001]->gpusettings[nvidia:0]]% ...;
[bright92->device[node001]]% mig show

<table>
<thead>
<tr>
<th>Node</th>
<th>GPU</th>
<th>MIG</th>
<th>Instance Name</th>
<th>Profile</th>
<th>Start</th>
<th>Size</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

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

Example

[bright92->device[node001]]% mig apply

<table>
<thead>
<tr>
<th>Node</th>
<th>GPU</th>
<th>MIG</th>
<th>Instance Name</th>
<th>Profile</th>
<th>Start</th>
<th>Size</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

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 with space separation). Using multiple --profile options is also possible.

Example

[bright92->device[node001]]% mig apply --profile 1g.5gb,2g.10gb

<table>
<thead>
<tr>
<th>Node</th>
<th>GPU</th>
<th>MIG</th>
<th>Instance Name</th>
<th>Profile</th>
<th>Start</th>
<th>Size</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

[bright92->device[node001]]% mig apply --profile 1g.5gb --profile 2g.10gb

<table>
<thead>
<tr>
<th>Node</th>
<th>GPU</th>
<th>MIG</th>
<th>Instance Name</th>
<th>Profile</th>
<th>Start</th>
<th>Size</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
• **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**

```
[bright92->device[node001]]% gpusettings; use nvidia:0
[bright92->device[node001]->gpusettings[nvidia:0]]% set migprofiles 7*1g.5gb; commit
[bright92->device[node001]->gpusettings[nvidia:0]]% ....
[bright92->device[node001]]% mig apply
```

<table>
<thead>
<tr>
<th>Node</th>
<th>GPU</th>
<th>MIG</th>
<th>Instance Name</th>
<th>Profile</th>
<th>Start</th>
<th>Size</th>
</tr>
</thead>
<tbody>
<tr>
<td>node001</td>
<td>0</td>
<td>7</td>
<td>-</td>
<td>1g.5gb</td>
<td>19</td>
<td>0</td>
</tr>
<tr>
<td>node001</td>
<td>0</td>
<td>7</td>
<td>0</td>
<td>1g.5gb</td>
<td>0</td>
<td>1</td>
</tr>
<tr>
<td>node001</td>
<td>0</td>
<td>8</td>
<td>-</td>
<td>1g.5gb</td>
<td>19</td>
<td>1</td>
</tr>
<tr>
<td>node001</td>
<td>0</td>
<td>8</td>
<td>0</td>
<td>1g.5gb</td>
<td>0</td>
<td>1</td>
</tr>
<tr>
<td>node001</td>
<td>0</td>
<td>9</td>
<td>-</td>
<td>1g.5gb</td>
<td>19</td>
<td>2</td>
</tr>
<tr>
<td>node001</td>
<td>0</td>
<td>9</td>
<td>0</td>
<td>1g.5gb</td>
<td>0</td>
<td>1</td>
</tr>
<tr>
<td>node001</td>
<td>0</td>
<td>10</td>
<td>0</td>
<td>1g.5gb</td>
<td>19</td>
<td>3</td>
</tr>
<tr>
<td>node001</td>
<td>0</td>
<td>10</td>
<td>0</td>
<td>1g.5gb</td>
<td>0</td>
<td>1</td>
</tr>
<tr>
<td>node001</td>
<td>0</td>
<td>11</td>
<td>-</td>
<td>1g.5gb</td>
<td>19</td>
<td>4</td>
</tr>
<tr>
<td>node001</td>
<td>0</td>
<td>11</td>
<td>0</td>
<td>1g.5gb</td>
<td>0</td>
<td>1</td>
</tr>
<tr>
<td>node001</td>
<td>0</td>
<td>12</td>
<td>-</td>
<td>1g.5gb</td>
<td>19</td>
<td>5</td>
</tr>
<tr>
<td>node001</td>
<td>0</td>
<td>12</td>
<td>0</td>
<td>1g.5gb</td>
<td>0</td>
<td>1</td>
</tr>
<tr>
<td>node001</td>
<td>0</td>
<td>13</td>
<td>-</td>
<td>1g.5gb</td>
<td>19</td>
<td>6</td>
</tr>
<tr>
<td>node001</td>
<td>0</td>
<td>13</td>
<td>0</td>
<td>1g.5gb</td>
<td>0</td>
<td>1</td>
</tr>
</tbody>
</table>

• **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**

```
[bright92->device[node001]]% gpusettings; use nvidia:0
[bright92->device[node001]->gpusettings[nvidia:0]]% set migprofiles 7g.40gb; commit
[bright92->device[node001]->gpusettings[nvidia:0]]% ....
[bright92->device[node001]]% mig apply
```

<table>
<thead>
<tr>
<th>Node</th>
<th>GPU</th>
<th>MIG</th>
<th>Instance Name</th>
<th>Profile</th>
<th>Start</th>
<th>Size</th>
</tr>
</thead>
<tbody>
<tr>
<td>node001</td>
<td>0</td>
<td>0</td>
<td>-</td>
<td>7g.40gb</td>
<td>0</td>
<td>8</td>
</tr>
<tr>
<td>node001</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>7g.40gb</td>
<td>4</td>
<td>7</td>
</tr>
</tbody>
</table>

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**

```
[bright92->device[node001]]% gpusettings; use nvidia:0
[bright92->device[node001]->gpusettings[nvidia:0]]% set migprofiles 2g.10gb; commit
[bright92->device[node001]->gpusettings[nvidia:0]]% ....
[bright92->device[node001]]% mig apply
```

<table>
<thead>
<tr>
<th>Node</th>
<th>GPU</th>
<th>MIG</th>
<th>Instance Name</th>
<th>Profile</th>
<th>Start</th>
<th>Size</th>
</tr>
</thead>
<tbody>
<tr>
<td>node001</td>
<td>0</td>
<td>5</td>
<td>-</td>
<td>2g.10gb</td>
<td>14</td>
<td>2</td>
</tr>
<tr>
<td>node001</td>
<td>0</td>
<td>5</td>
<td>0</td>
<td>2g.10gb</td>
<td>1</td>
<td>2</td>
</tr>
</tbody>
</table>
• **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**

```
[bright92->device[node001]]% gpusettings; use nvidia:0
[bright92->device[node001]->gpusettings[nvidia:0]]% set migprofiles 2g.10gb:0; commit
[bright92->device[node001]->gpusettings[nvidia:0]]% ....
[bright92->device[node001]]% mig apply
```

<table>
<thead>
<tr>
<th>Node</th>
<th>GPU</th>
<th>MIG</th>
<th>Instance Name</th>
<th>Profile</th>
<th>Start</th>
<th>Size</th>
</tr>
</thead>
<tbody>
<tr>
<td>node001</td>
<td>0</td>
<td>5</td>
<td>-</td>
<td>2g.10gb</td>
<td>14</td>
<td>4</td>
</tr>
<tr>
<td>node001</td>
<td>0</td>
<td>5</td>
<td>0</td>
<td>1c.2g.10gb</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>node001</td>
<td>0</td>
<td>5</td>
<td>1</td>
<td>1c.2g.10gb</td>
<td>0</td>
<td>1</td>
</tr>
</tbody>
</table>

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**

```
[bright92->device[node001]]% gpusettings; use nvidia:0
[bright92->device[node001]->gpusettings[nvidia:0]]% set migprofiles 2g.10gb:0:0; commit
[bright92->device[node001]->gpusettings[nvidia:0]]% ....
[bright92->device[node001]]% mig apply
```

<table>
<thead>
<tr>
<th>Node</th>
<th>GPU</th>
<th>MIG</th>
<th>Instance Name</th>
<th>Profile</th>
<th>Start</th>
<th>Size</th>
</tr>
</thead>
<tbody>
<tr>
<td>node001</td>
<td>0</td>
<td>5</td>
<td>-</td>
<td>2g.10gb</td>
<td>14</td>
<td>4</td>
</tr>
<tr>
<td>node001</td>
<td>0</td>
<td>5</td>
<td>0</td>
<td>1c.2g.10gb</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>node001</td>
<td>0</td>
<td>5</td>
<td>1</td>
<td>1c.2g.10gb</td>
<td>0</td>
<td>1</td>
</tr>
</tbody>
</table>

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:

<table>
<thead>
<tr>
<th>compact colon form</th>
<th>expanded form</th>
</tr>
</thead>
<tbody>
<tr>
<td>:0</td>
<td>1c.&lt;profile&gt;</td>
</tr>
<tr>
<td>:1</td>
<td>2c.&lt;profile&gt;</td>
</tr>
<tr>
<td>...</td>
<td>...</td>
</tr>
<tr>
<td>:N-1</td>
<td>Nc.&lt;profile&gt;  or &lt;profile&gt;</td>
</tr>
</tbody>
</table>
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:

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

```
[bright92->device[node001]]% gpusettings; use nvidia:0
[bright92->device[node001]->gpusettings[nvidia:0]]% set migprofiles 3*2g.10gb:0:0; commit
[bright92->device[node001]->gpusettings[nvidia:0]]% ...;
[bright92->device[node001]]% mig apply
```

<table>
<thead>
<tr>
<th>Node</th>
<th>GPU</th>
<th>MIG</th>
<th>Instance Name</th>
<th>Profile</th>
<th>Start</th>
<th>Size</th>
</tr>
</thead>
<tbody>
<tr>
<td>node001</td>
<td>0</td>
<td>3</td>
<td>-</td>
<td>2g.10gb</td>
<td>14</td>
<td>0</td>
</tr>
<tr>
<td>node001</td>
<td>0</td>
<td>3</td>
<td>0</td>
<td>1c.2g.10gb</td>
<td>0</td>
<td>1</td>
</tr>
<tr>
<td>node001</td>
<td>0</td>
<td>3</td>
<td>1</td>
<td>1c.2g.10gb</td>
<td>0</td>
<td>1</td>
</tr>
<tr>
<td>node001</td>
<td>0</td>
<td>4</td>
<td>-</td>
<td>2g.10gb</td>
<td>14</td>
<td>2</td>
</tr>
<tr>
<td>node001</td>
<td>0</td>
<td>4</td>
<td>0</td>
<td>1c.2g.10gb</td>
<td>0</td>
<td>1</td>
</tr>
<tr>
<td>node001</td>
<td>0</td>
<td>4</td>
<td>1</td>
<td>1c.2g.10gb</td>
<td>0</td>
<td>1</td>
</tr>
<tr>
<td>node001</td>
<td>0</td>
<td>5</td>
<td>-</td>
<td>2g.10gb</td>
<td>14</td>
<td>4</td>
</tr>
<tr>
<td>node001</td>
<td>0</td>
<td>5</td>
<td>0</td>
<td>1c.2g.10gb</td>
<td>0</td>
<td>1</td>
</tr>
<tr>
<td>node001</td>
<td>0</td>
<td>5</td>
<td>1</td>
<td>1c.2g.10gb</td>
<td>0</td>
<td>1</td>
</tr>
</tbody>
</table>

• 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:

```
[bright92->device[node001]]% gpusettings; use nvidia:0
[bright92->device[node001]->gpusettings[nvidia:0]]% set migprofiles 2*1g.5gb 2g.10gb; commit
[bright92->device[node001]->gpusettings[nvidia:0]]% show
Parameter | Value
----------|-------------------------------------------------
MIG profiles | 2*1g.5gb,2g.10gb
```

```
[bright92->device[node001]]% ...;
[bright92->device*[node001*]]% mig apply
```

<table>
<thead>
<tr>
<th>Node</th>
<th>GPU</th>
<th>MIG</th>
<th>Instance Name</th>
<th>Profile</th>
<th>Start</th>
<th>Size</th>
</tr>
</thead>
<tbody>
<tr>
<td>node001</td>
<td>0</td>
<td>3</td>
<td>-</td>
<td>2g.10gb</td>
<td>14</td>
<td>0</td>
</tr>
<tr>
<td>node001</td>
<td>0</td>
<td>3</td>
<td>0</td>
<td>2g.10gb</td>
<td>1</td>
<td>2</td>
</tr>
<tr>
<td>node001</td>
<td>0</td>
<td>11</td>
<td>-</td>
<td>1g.5gb</td>
<td>19</td>
<td>4</td>
</tr>
<tr>
<td>node001</td>
<td>0</td>
<td>11</td>
<td>0</td>
<td>1g.5gb</td>
<td>0</td>
<td>1</td>
</tr>
<tr>
<td>node001</td>
<td>0</td>
<td>13</td>
<td>-</td>
<td>1g.5gb</td>
<td>19</td>
<td>6</td>
</tr>
<tr>
<td>node001</td>
<td>0</td>
<td>13</td>
<td>0</td>
<td>1g.5gb</td>
<td>0</td>
<td>1</td>
</tr>
</tbody>
</table>
• 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:

\[2 \times 1g.5gb \quad 2g.10gb\]

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:

\[5 \times 1g.5gb, 2g.10gb\]

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\times 1g.5gb, 2g.10gb allocates the 5 slices of the 1g instance and the 2 slices of the 2g instance. But
- 6\times 1g.5gb, 2g.10gb allocates the 6 slices of the 1g instance and none of the 2g instance.
- 7\times 1g.5gb, 2g.10gb allocates the 7 slices of the 1g instance and none of the 2g instance.
- 70\times 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:

```bash
#!/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:
Configuring The Cluster

### Example

```
[root@bright92]# cmsh -c "device; format hostname:15, status:15; list"
hostname (key) status
----------------- --------------
bright92 [ 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:

```
[root@bright92]# cmsh
[bright92]# device use node001
[bright92->device[node001]]% get customremoteconsolescript; get customremoteconsolescriptargument

[bright92->device[node001]]% set customremoteconsolescript /root/kvmaccesshack
[bright92->device[node001]]% set customremoteconsolescriptargument 1
[bright92->device[node001]]% rconsole
```

KVM console access session using the 1 argument option is displayed

In Bright View, the corresponding clickpaths to access these script settings are:

- Devices → Nodes ↓ Edit → Settings → Custom remote console script
- 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:

<table>
<thead>
<tr>
<th>Type of script</th>
<th>Script directory path on node</th>
</tr>
</thead>
<tbody>
<tr>
<td>local</td>
<td>/cm/local/apps/cmd/scripts/sysinfo/local/</td>
</tr>
<tr>
<td>director</td>
<td>/cm/local/apps/cmd/scripts/sysinfo/director/</td>
</tr>
<tr>
<td>head</td>
<td>/cm/local/apps/cmd/scripts/sysinfo/head/</td>
</tr>
</tbody>
</table>

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
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**

```bash
[root@bright92 ~]# 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**

```bash
[root@bright92 ~]# /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": "bright92",
"bright92 UUID": "6733d33a-2933-41ea-aa3c-b218e794c8b9"
}
```

**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@bright92 ~]# cp -r /cm/local/apps/cmd/scripts/sysinfo/head/outputscript.sh /cm/images/default-image/cm/local/apps/cmd/scripts/sysinfo/local/
[root@bright92 ~]# cmsh
[bright92]# device
[bright92->device]# reboot node001

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

[bright92->device]# sysinfo node001 --update
[bright92->device]# sysinfo node001

```
Age       11s
CMDaemon running on node001
script running on node001
node001 UUID 3b1c7973-07ef-419f-9324-4edf99690d5
script path is /cm/local/apps/cmd/scripts/sysinfo/local/outputscript.sh
```

```
CMDaemon running on node001
script running on bright92
bright92 UUID 6733d33a-2933-41ea-aa3c-b218e78d4b9
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 Configuring Composable Infrastructure

3.15.1 Why Use Composable Infrastructure?

Composable infrastructure is infrastructure that is made up of hardware resources, designed at a hardware level to be *composed* (built up) by software.

To understand why composable infrastructure is useful in a cluster, it is helpful to first consider the issues with using non-composable infrastructure in a cluster.
Non-composable Infrastructure Issues
For any cluster, composable or non-composable, a cluster administrator wants to optimize the use of the budget allocated for the cluster. To do this, the cluster administrator would typically try to minimize resources, such as storage, memory, networking, or GPUs, going unused.

In a traditional, non-composable, cluster, resources are often spread non-uniformly throughout the cluster infrastructure, and often optimized around particular nodes. For such a cluster, the cluster administrator therefore usually inventorizes the hardware resources, and organizes the resources in some way. For example, in the cluster manager the cluster administrator may organize nodes:

- with categories (section 2.1.3). For example, there might be a category of nodes with 4 GPUs.
- with node groups (section 2.1.4). For example, there might be a group of nodes to be used by the Aeronautics Department.
- according to their storage properties (could be a category or a group). For example, nodes having large amounts of storage.
- according to their connection properties (could be a category or a group). For example, nodes in a particular room with InfiniBand for extra bandwidth.

The allocation of resources is then planned out for users and groups of users according to need. As needs change, the original plan may need to be modified, and resources reallocated.

Reallocation of resources at the software level is usually not an issue. For example, in the case of the Space Force Department getting allocated some nodes that were originally allocated to the Aeronautics Department—that can simply be done with re-categorization.

However, it often turns out that the optimal way to reorganize the resources in the cluster also requires reorganizing the underlying physical hardware, and knowing exactly what resources are associated with the underlying nodes. For example, the Space Force Department might need several GPUs, but the node given to them by the Aeronautics Department only has one, while the Aeronautics Department keeps the only other GPU node, which has 4 GPUs, because it has a huge amount of storage. Eventually, the problem is fixed by opening up the Aeronautics Department node, and moving some GPU cards and hard drives over to the Space Force Department node. This is also not usually much of an issue, if it happens once.

The trouble is, that cluster hardware can get reorganized periodically. Also, in general, if physical hardware changes are done regularly to nodes during reorganization, then it can make a cluster harder to manage over time (figure 3.25):

Because of the understandable reluctance by cluster administrators to carry out hardware reorganization and reallocation, resource use on a traditional cluster has a tendency to remain static, and as a result, users over time may find that their performance needs are no longer met by the original infrastructure.

How Is Composable Infrastructure An Improvement On Non-composable Infrastructure?
Resource allocation from a common pool: In a cluster that uses composable infrastructure, resource allocation is dealt with by putting all the resources into a common pool. Typically the hardware for the pool is specialized, with, for example, storage provided by uniform, dedicated storage nodes, and GPUs provided by uniform, dedicated nodes with GPUs.

When resources are needed, they are allocated from the common pool. This is convenient because the cluster administrator no longer has to think as much about the physical infrastructure underlying the cluster. Each resource has been abstracted away from the underlying hardware, and placed into the pool for that type of resource, so that the resource is now fungible.

The abstraction means that resources composition can transcend the conventional physical limits. For example, if there are 8 GPU cards on the cluster, then even if there is no machine with 8 physical slots
3.15 Configuring Composable Infrastructure

Figure 3.25: An example of hardware configuration that may be hard to manage to accept GPU cards, the 8 GPUs can still be used by one job running on a machine that has requested the use of 8 GPU resources from the common pool.

**Resource communication:** Having each resource in a common pool may be convenient, but there may be a concern about the performance of the underlying hardware when a connection is made to use that resource. A pooled resource might be expected to have a communication penalty associated with it.

The usual way that composable infrastructure deals with hardware communication is by providing a high performance fabric connectivity hardware (cabling and switches) to bypass traditional TCP/IP networking. The dedicated fabric reduces CPU load, reduces latency, and allows a higher bandwidth. Among the fabric technologies in use for composable infrastructure are InfiniBand, PCIe, and Gen-Z.

In composable infrastructure, the composable resources and their interconnects are thus composable. This means that, to meet the needs of users, instead of requiring the cluster administrator to physically carry out re-cabling and move hardware components around as is the case with non-composable infrastructure, it is possible to reorganize the composable resources of the cluster virtually via the cluster manager.

### 3.15.2 GigaIO Composable Infrastructure

The cluster manager supports the GigaIO infrastructure components. GigaIO uses a PCIe fabric technology, FabreX™, as the interconnect for the transport layer hardware and protocol.

**GigaIO Composable Infrastructure Components**

The following hardware components are part of a GigaIO-based composable infrastructure:

- Resource boxes: Containers with GPUs, storage, or other resources that connect via PCIe.
- Nodes: Nodes that have PCIe extender cards placed in them to connect to the PCIe fabric.
- Fabric switches: Dedicated switches that use PCIe interconnects to connect to the resource boxes and to the nodes.
- Fiber links: PCIe fiber interconnects used from the fabric switches, going to the resource boxes, to the nodes, or to other switches.
3.15.3 The Cluster Manager Composable Infrastructure Integration Concepts

The cluster manager integration of composable infrastructure uses the following concepts:

- A **binding** connects resources to host nodes, and is allocated by the cluster administrator. Bindings are virtual—the software representation is only concerned with the end points, and not the physical communication path.

- A **link** in a switch represents the cable between two switches that links the switches. The link can be:
  - a cascade link, which is part of the path from a node to a resource box,
  - a cross link, which is part of the path from one node to another node for computational needs
  - an NT link (non-transparent link), which is part of the path used for communicating node-to-node without exposing the data that each node accesses in its resources.

In composable infrastructure terminology in general, each link has two ends. If the connection is a node-to-resource cascade link, then one link end, the DSP (DownStream Port) end, connects to the USP (UpStream Port) of the switch. The other end of the link is then the USP for the DSP of the switch. The directions, up and down, indicated by the labels, have nothing to do with actual direction of packet data flow, which is bidirectional. The label is just associated with ownership, and based on the node port(s) of the switch being the USP end, and the resource port(s) of the switch being the DSP end.

In the case of nodes that simply rely on fast interconnects to each other via switches, and do not have a resource box destination, both ends of the link are designated as USP.

- A **zone** can be thought of as being like a TCP/IP subnet. Just like for TCP/IP subnets, the transport layer packets that run over PCIe can be organized by the switch into separate zones, for speed and security. The packets from one zone are dealt with in their zone only, and only nodes and resources that are in the same zone can have a binding between them.

  The **NTZone** (Non-Transparent Zone) is a special zone for node-to-node communication, effectively crossing zones for the nodes, but with the zone communications (which are transparent within a zone) still remaining inaccessible to other zones.

- A **topology** is how the physical interconnects are laid out across the nodes, switches, and resource boxes. Topologies are only available from a predefined selection. After selecting the topology, the bindings are defined on top of the topology by the cluster administrator, to complete the fabric configuration.

  For example, a topology with one node, one switch, and one resource box, would have no links, would have one or more USPs on the node side of the switch, and would have one or more DSPs on the resource side the switch. Setting a complete node-to-resource binding path in this example, would require a binding from the node to a USP on the switch, and another binding from the DSP on the switch to the resource box. This example is described more fully on page 162.

- A **fabric** is a configuration made up of a particular topology and binding. At a hardware level it is composed of a topology of fabric switches and resource boxes. There can be several fabrics in a cluster.

3.15.4 Standing Up A Bright Cluster With The GigaIO Fabric

This section covers the steps for standing up a Bright cluster with GigaIO composable infrastructure. The steps are:

1. **Cable as per required topology:** Once the devices have been installed in the rack(s), all devices must be cabled to match the desired topology.
2. **Install the head node:** The head node is installed using a Bright ISO based on CentOS 7 or Ubuntu 20.04. At the time of writing of this section (June 2022), CentOS 7 and Ubuntu 20.04 are the only validated Linux distributions.

3. **Define fabric switches:** Once the head node has an UP status, and is running, then the fabric switches can be defined using Bright View or cmsh. The minimum required configuration is the hostname, IP address, username and password.

4. **Define nodes (hosts):** Using Bright View or cmsh, nodes are defined that should be part of the composable infrastructure fabric.

5. **Define resource boxes:** Using Bright View or cmsh, resource boxes are defined that comprise resources to be used for composing nodes (hosts).

6. **Power on the fabric switches:** The fabric switches that are used in managing the composability of resources are powered on.

7. **Configure topology:** The desired topology is configured in Bright and the settings are propagated to the switches. To do that:
   - Set fabric switches: The relevant switches defined in step 3 are selected.
   - Set hosts: The relevant hosts defined in step 4 are selected.
   - Set resource boxes: The resource boxes defined in step 5 are selected.
   - Apply topology: The desired topology changes are propagated to the fabric switches.

8. **Create GigaIO software image:** A software image is created that includes the GigaIO host software stack using the `cm-create-gigaio-image` tool. The software image is assigned to the appropriate nodes or node categories.

**GigaIO host software stack installation:** For the GigaIO composable infrastructure management to work, the nodes to which resources can be allocated must run the GigaIO FabreX™ host software stack. The FabreX software stack includes a custom kernel and additional utilities. The software stack must be installed into the software image used by the nodes.

The cluster manager provides a utility called `cm-create-gigaio-image` that allows users to install the FabreX software stack into a target software image, and also sets the correct kernel version for the software image.

The FabreX software stack must be obtained via GigaIO support. It is usually provided as a compressed tar archive. The path to the archive is passed an argument to `cm-create-gigaio-image` as the source of packages. An alternative is to specify a directory to where the archive has already been extracted.

The help text for `cm-create-gigaio-image` is:

```
[root@gigaio ~]# cm-create-gigaio-image -h
usage: cm-create-gigaio-image [-h] [--clonefrom CLONEFROM] [--verbose] packages targetimage

Create gigaio software image

positional arguments:
  packages     Path to directory or archive containing gigaio packages
  targetimage  Name of target software image

optional arguments:
  -h, --help    show this help message and exit
```
154 Configuring The Cluster

--clonefrom CLONEFROM

Name of software image to clone from

--verbose

Enable verbose logging

Some example of its use follow:

1. The FabreX software stack can be installed from the specified compressed archive to an already-existing image giga-image-v25-r4, with:

   cm-create-gigaio-image --verbose /root/FabreX_v2.5.0r4_CentOS7.tar.bz2 gigaio-image-v25-r4

   For a Ubuntu 20.04 distribution, the package is FabreX_v2.5.0r4_Ubuntu20.tar.bz2.

2. The image default-image is cloned to the target image gigaio-image-v25-r4, installed with the FabreX software stack from the specified directory, and the correct kernel version for the target image is set, with:

   export IGNORE_POST_INSTALL_FAILURES=1
   cm-create-gigaio-image --clonefrom default-image /root/FabreX_v2.5.0r4_CentOS7 gigaio-image-v25-r4

   The IGNORE_POST_INSTALL_FAILURES environment variable is necessary at the time of writing (June 2022), because not all GigaIO packages currently install cleanly into a software image. This flag is expected to be not needed in a future version, probably v2.6.

9. Boot nodes: After the new image has been created, nodes can be set to use the GigaIO software image:

   Example

   root@gigaio2:~# cmsh
   [bright92]% category use default
   [bright92->category[default]]% set softwareimage gigaio-image-v25-r4
   [bright92->category[default]]% commit
   [bright92->category[default]]% device power reset -c default
   drac0 .................... [ RESET ] node001
   drac0 .................... [ RESET ] node002
   drac0 .................... [ RESET ] node003
   drac0 .................... [ RESET ] node004

10. Verify that the nodes are part of the fabric: The fabrex-status.sh utility can be used to check the status of the fabric connectivity and ensure that the host is part of the composable infrastructure.

   An example that runs the check on all compute nodes follows:

   [root@gigaio ]# pdsh -g computenode "fabrex-status.sh | grep -A 3 PNUM"

   node004: PNUM STATE PCCS LLINK POLL PSTATE PLINK EPOCH FLID PEER ID
   node004: 0 rdy cfg up slow rdy up 1 0x00000001 node001
   node004: 1 rdy cfg up slow rdy up 1 0x00000002 node002
   node004: 2 rdy cfg up slow rdy up 1 0x00000003 node003
   node002: PNUM STATE PCCS LLINK POLL PSTATE PLINK EPOCH FLID PEER ID
   node002: 0 rdy cfg up slow rdy up 1 0x00000001 node001
   node002: 1 rdy cfg up slow rdy up 1 0x00000003 node003
   node002: 2 rdy cfg up slow rdy up 1 0x00000004 node004
   node001: PNUM STATE PCCS LLINK POLL PSTATE PLINK EPOCH FLID PEER ID
   node001: 0 rdy cfg up slow rdy up 1 0x00000003 node003
   node001: 1 rdy cfg up slow rdy up 1 0x00000004 node004
   node001: 2 rdy cfg up slow rdy up 1 0x00000002 node002
   node003: PNUM STATE PCCS LLINK POLL PSTATE PLINK EPOCH FLID PEER ID
   node003: 0 rdy cfg up slow rdy up 1 0x00000001 node001
   node003: 1 rdy cfg up slow rdy up 1 0x00000002 node002
   node003: 2 rdy cfg up slow rdy up 1 0x00000004 node004
Setting Up The Fabric Switches And Resource Boxes In Composable Infrastructure As Bright Objects

In the cluster manager, the fabric switches and resource boxes of composable infrastructure can be added via the `add` command (page 28) within `device` mode, similarly to how it is done for regular nodes (section 5.7) and other objects in the cluster manager.

Example

```
[head->device] % add fabricswitch fswitch001
[head->device*[fswitch001*]] % show
Parameter       Value
Device height
Device position
Hostname       fswitch001
Revision
Type           FabricSwitch
Tag             00000000a000
Mac          00:00:00:00:00:00
Activation    N/A
Rack
Container index  0
Vendor
Ports          24
Lanes per port  4
Power control  none
Custom power script
Custom power script argument
Power distribution units
Switch ports
Network
Ip             0.0.0.0
User name
Password     < not set >
Userdefined1
Userdefined2
User defined resources
Custom ping script
Custom ping script argument
Partition       base
Notes          <0B>
```

Similarly for a resource box:

```
[head->device] % add fabricresourcebox frbox001
[head->device*[my-io01*]] % show
Parameter       Value
Device height
Device position
Hostname       frbox001
Revision
Type           FabricResourceBox
Tag             00000000a000
Mac          00:00:00:00:00:00
Activation    N/A
Rack
```
In general, adding objects that have been designed for integration into the cluster manager means that the properties of such objects can be managed. Once their properties are correctly set up, then the objects can be managed by the cluster manager. So, for composable infrastructure objects, this means that after the fabric switches and resource boxes have been added as objects, infrastructure composition can then be managed via the cluster manager (section 3.15.5).

Properties, such as the IP address, username, and password can be set for the composable infrastructure objects just added, as follows:

Example

```
[head->device[fswitch001]]% set ip 10.141.1.1
[head->device[fswitch001]]% set mac 00:22:3D:7D:00:B5
[head->device[fswitch001]]% set username admin@example.com
[head->device[fswitch001]]% set password password1
[head->device[fswitch001]]% set network internalnet
[head->device[fswitch001]]% set vendor gigaio
[head->device[fswitch001]]% commit
[head->device[fswitch001]]% exit

[head->device[fswitch002]]% set ip 10.141.1.2
[head->device[fswitch002]]% set mac 00:22:3D:7D:00:c5
[head->device[fswitch002]]% commit
[head->device[fswitch002]]% exit

[head->device]% clone fswitch001 fswitch002 # fabric switch2 conveniently cloned
```

If the MAC addresses have been set correctly, then the DHCP server on the Bright head node can assign IP address information to the GigaIO switches and resource boxes. A powercycle of all GigaIO switches and resource boxes triggers them to obtain a DHCP lease.

The device status command initially shows the newly-added devices as being down:

Example
3.15 Configuring Composable Infrastructure

After the powercycle, they are shown as being up:

Example

```bash
[head->device]% device status
frbox001 ................. [ UP   ]
fswitch001 ............... [ UP   ]
fswitch002 ............... [ UP   ]
head ..................... [ UP   ]
node001 .................. [ UP   ]
node002 .................. [ UP   ]
node003 .................. [ UP   ]
node004 .................. [ UP   ]
```

The cluster manager can now use the fabric switches and resource boxes for composing the composable infrastructure.

3.15.5 Using The Cluster Manager For Composable Infrastructure With GigaIO

This section assumes a GigaIO host software stack installation has been carried out, and that the fabric switches and resource boxes have been set up and are ready for composition (section 3.15.4).

Setting Up A Fabric Configuration

The cmsh modes tree has the following layout for composable infrastructure:

```
|-- partition
   `-- fabrics
       |-- bindings
       `-- topology
```

A fabric configuration can be specified by dropping into `fabrics` mode. This is a submode under the main `partition` mode. As is usual when working with cmsh objects (section 2.5.3), a new fabric configuration object can be built using the `add` command. The bindings and topology for that fabric can then be set:

Example

```bash
[head->partition[base]->fabrics]% add gigaio-2
[head->partition*[base*]->fabrics*[gigaio-2*]]% show
 Parameter                      Value
--------------------------------- ------------------------------------------------
Name                             gigaio-2
Revision                         
Binding method                   PATCH
Unbind all after topology apply  yes
Topology                        
Switches                         
```
Nodes
Resource boxes
[head->partition*[base*]->fabrics*[gigaio-2*]]% topology
[head->partition*[base*]->fabrics*[gigaio-2*]->topology]% show
Parameter Value
Name 158 Configuring The Cluster
Revision
Template <4B>
Management switch undefined
[head->partition*[base*]->fabrics*[gigaio-2*]->topology]% ..; bindings
[head->partition*[base*]->fabrics*[gigaio-2*]->bindings]% show
[head->partition*[base*]->fabrics*[gigaio-2*]->bindings]%

Setting Up A Topology

Topology selection: To set a topology, a selection must be made in topology mode. A topology cannot be added, it can only be selected from the available topologies. The available command queries the GigaIO switch firmware and displays a summary of the available predefined topologies, while the select command allows tab-prompting to suggest the available topologies:

Example

[head->partition*[base*]->fabrics*[gigaio-2*]->topology]% available
Sources Name Switches Zones Links Hosts DSPs
----------------------- ---------------- -------- -------- -------- -------- --------
f-switch01..f-switch30 15S-12x24-1 15 7 27 12 24
f-switch01..f-switch30 1S-2x4-1 1 2 0 2 4
f-switch01..f-switch30 1S-6x0-1 1 2 0 0 0
f-switch01..f-switch30 2S-4x4-1 2 2 2 4 4
f-switch01..f-switch30 2S-4x4-3 2 2 4 4 4
f-switch01..f-switch30 30S-24x48-1 30 13 54 24 48
f-switch01..f-switch30 3S-12x0-1 3 2 3 12 0
f-switch01..f-switch30 7S-6x12-1 7 4 12 6 12

[head->partition*[base*]->fabrics*[gigaio-2*]->topology]% select <tab><tab>
1S-2x4-1
[head->partition*[base*]->fabrics*[gigaio-2*]->topology]% select <tab><tab>
1S-2x4-1 1S-6x0-1 2S-4x4-1 2S-4x4-3 30S-24x48-1 3S-12x0-1 7S-6x12-1
[head->partition*[base*]->fabrics*[gigaio-2*]->topology]% select 1S-2x4-1
[head->partition*[base*]->fabrics*[gigaio-2*]->topology]% show
Parameter Value
Name 1S-2x4-1
Revision
Template <283B>
Management switch Switch1
Zone 0 NT
Zone 1 zone1
Switch 1
Host switch 1, port 1
Host switch 1, port 5
DSP switch 1, port 9
DSP switch 1, port 13
DSP switch 1, port 17
DSP switch 1, port 21
The available query provides the cluster manager with what is available in the pool for each topology, as well as the number of switches, links, and zones. The switches have defined host ports, and defined resource box ports. The cluster administrator has to set hosts for the USPs to connect to, and resource boxes (if any) for the DSPs to connect to. Links already have their names set based on the topology definition. Zones are also predefined, based on the topology definition.

The names for the predefined topologies are based on the numbers in the format:

<switches>-<nodes> x <resource boxes>-<serial>

So, in the last session, the topology is a single switch, with 2 nodes connected to the switch, and 4 resource boxes connected to the switch. The serial number is an extra identifier label, to distinguish between otherwise identical topologies that are connected up in a different way.

The software representation of the topology defined in the object must match the hardware, or the fabric interconnects will fail when the configuration is applied.

**Topology parameter settings:** The topology representation can be set by defining the values for:

- the name of the fabric switch device
- the hosts that the switches connect to
- the resource boxes that the switches connect to

This does not specify the bindings. Bindings are how the devices connected to the switch are bound to each other. Only after the topology is set, can the bindings be set.

If selecting the 2s-4x4-3 topology, a session to set the parameter values of the topology might run as follows:

**Example**

```bash
[head->partition[base]->fabrics]% topology gigaio-1
[head->partition[base]->fabrics[GigaIO-1]->topology]% select 2s-4x4-3
[head->partition*[base*]->fabrics*[GigaIO-1*]->topology]% show
```

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>Name</td>
<td>2S-4x4-3</td>
</tr>
<tr>
<td>Revision</td>
<td>&lt;621B&gt;</td>
</tr>
<tr>
<td>Management switch</td>
<td>Switch1</td>
</tr>
<tr>
<td>Zone 0</td>
<td>NT</td>
</tr>
<tr>
<td>Zone 1</td>
<td>zone1</td>
</tr>
<tr>
<td>Switch 1</td>
<td></td>
</tr>
<tr>
<td>Switch 2</td>
<td></td>
</tr>
<tr>
<td>s1p17dsp</td>
<td>s2p1usp</td>
</tr>
<tr>
<td>s1p19dsp</td>
<td>s2p3usp</td>
</tr>
<tr>
<td>s1p21dsp</td>
<td>s2p5usp</td>
</tr>
<tr>
<td>s1p23dsp</td>
<td>s2p7usp</td>
</tr>
<tr>
<td>Host switch 1, port 1</td>
<td></td>
</tr>
<tr>
<td>Host switch 1, port 5</td>
<td></td>
</tr>
<tr>
<td>Host switch 1, port 9</td>
<td></td>
</tr>
<tr>
<td>Host switch 1, port 13</td>
<td></td>
</tr>
<tr>
<td>DSP switch 2, port 9</td>
<td></td>
</tr>
<tr>
<td>DSP switch 2, port 13</td>
<td></td>
</tr>
<tr>
<td>DSP switch 2, port 17</td>
<td></td>
</tr>
<tr>
<td>DSP switch 2, port 21</td>
<td></td>
</tr>
</tbody>
</table>
The 2 switches, the 4 host node (USP) connections, and the 4 resource box port (DSP) connections, all need to be defined.

The host nodes are the usual regular nodes, so 4 can be allocated to the USP connections from existing node objects.

The switches are fabric switches, and 2 need to be represented. The switch objects can be added in device mode, if they do not already exist (page 155). Similarly, the resource box objects can be added in device mode, if they do not already exist.

The definitions are carried out with the set command:

**Example**

```bash
[head->partition*[base*]->fabrics*[GigaIO-1*]->topology*]% set host node001..node004
[head->partition*[base*]->fabrics*[GigaIO-1*]->topology*]% set switch f-switch01..f-switch02
[head->partition*[base*]->fabrics*[GigaIO-1*]->topology*]% set dsp io-01..io-04
[head->partition*[base*]->fabrics*[GigaIO-1*]->topology*]% commit
[head->partition*[base*]->fabrics*[GigaIO-1*]->topology*]% show
```

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>Name</td>
<td>2S-4x4-3</td>
</tr>
<tr>
<td>Revision</td>
<td></td>
</tr>
<tr>
<td>Template</td>
<td>&lt;621B&gt;</td>
</tr>
<tr>
<td>Management switch</td>
<td>Switch1 (f-switch01)</td>
</tr>
<tr>
<td>Zone 0</td>
<td>NT</td>
</tr>
<tr>
<td>Zone 1</td>
<td>zone1</td>
</tr>
<tr>
<td>Switch 1</td>
<td>f-switch01</td>
</tr>
<tr>
<td>Switch 2</td>
<td>f-switch02</td>
</tr>
<tr>
<td>s1p17dsp &lt;-&gt; s2p1usp</td>
<td>s1s2lk1</td>
</tr>
<tr>
<td>s1p19dsp &lt;-&gt; s2p3usp</td>
<td>s1s2lk2</td>
</tr>
<tr>
<td>s1p21 dsp &lt;-&gt; s2p5usp</td>
<td>s1s2lk3</td>
</tr>
<tr>
<td>s1p23 dsp &lt;-&gt; s2p7usp</td>
<td>s1s2lk4</td>
</tr>
<tr>
<td>Host switch 1, port 1</td>
<td>node001</td>
</tr>
<tr>
<td>Host switch 1, port 5</td>
<td>node002</td>
</tr>
<tr>
<td>Host switch 1, port 9</td>
<td>node003</td>
</tr>
<tr>
<td>Host switch 1, port 13</td>
<td>node004</td>
</tr>
<tr>
<td>DSP switch 2, port 9</td>
<td>io-01</td>
</tr>
<tr>
<td>DSP switch 2, port 13</td>
<td>io-02</td>
</tr>
<tr>
<td>DSP switch 2, port 17</td>
<td>io-03</td>
</tr>
<tr>
<td>DSP switch 2, port 21</td>
<td>io-04</td>
</tr>
</tbody>
</table>

When the commit command is run the topology object is stored in the cluster manager with the new settings, and the show command shows the stored settings. However, the new settings are not applied to the running composable system until the apply command is run.

**The check command in the topology submode:** It is possible to make mistakes in cabling configuration. The topology can be checked with the check command:

**Example**

```bash
[head->partition*[base*]->fabrics*[GigaIO-1*]->topology*]% check
```

<table>
<thead>
<tr>
<th>Hostname</th>
<th>UUID</th>
<th>Good</th>
<th>Information</th>
</tr>
</thead>
<tbody>
<tr>
<td>node001</td>
<td>4c4c4544-0048-4610-904a-b6c04f523532</td>
<td>yes</td>
<td></td>
</tr>
<tr>
<td>node002</td>
<td>4c4c4544-0044-3410-3035-c2c04f44b32</td>
<td>yes</td>
<td></td>
</tr>
<tr>
<td>node003</td>
<td>4c4c4544-0053-4810-8057-c4c04f434832</td>
<td>yes</td>
<td></td>
</tr>
<tr>
<td>node004</td>
<td>4c4c4544-0053-4a10-8056-c4c04f434835</td>
<td>yes</td>
<td></td>
</tr>
</tbody>
</table>
It may be helpful for troubleshooting to know that the field of the UUID after the last hyphen is the MAC address of the node.

**Verifying the live state with the `showlive` command in the topology submode:** The cluster administrator can verify if the changes have actually been applied by running the `showlive` command to see the live state:

```
[head->partition[base]->fabrics[GigaIO-1]->topology]% showlive
{
    "manager": "f-switch01",
    "name": "1S-2x4-1",
    "settings": {
        "s1p13dsp": "io-02",
        "s1p17dsp": "io-03",
        "s1p1host": "node001",
        "s1p21dsp": "io-04",
        "s1p5host": "node002",
        "s1p9dsp": "io-01",
        "switch1": "f-switch01",
        "zone0": "NT",
        "zone1": "zone1"
    }
}
[head->partition[base]->fabrics[GigaIO-1]->topology]% apply
```

Some devices are UP, power them off or use `--force`

```
io-01, io-02, io-03, io-04, node001
```

After a reboot of affected nodes that are up, the live changes for the topology should show up:

```
[head->partition[base]->fabrics[GigaIO-1]->topology]% showlive
{
    "manager": "f-switch01",
    "name": "2S-4x4-3",
    "settings": {
        "s1p13host": "node004",
        "s1p17dsp": "s1s2lk1",
        "s1p19dsp": "s1s2lk2",
        "s1p1host": "node001",
        "s1p21dsp": "s1s2lk3",
        "s1p23dsp": "s1s2lk4",
        "s1p5host": "node002",
        "s1p9host": "node003",
        "s2p13dsp": "io-02",
        "s2p17dsp": "io-03",
        "s2p21dsp": "io-04",
        "s2p9dsp": "io-01",
        "switch1": "f-switch01",
        "switch2": "f-switch02",
        "zone0": "NT",
        "zone1": "zone1"
    }
}
```

The reason the `apply` command must be run explicitly is because a changed topology pushes drastic changes to the fabric, which, if incorrect, can cause the fabric to cease working. Separating the topology application from the topology representation helps prevent a cluster administrator from automatically carrying out the change.
Setting up dynamic nodes: Nodes in topology mode can be specified by the administrator to be part of a static or dynamic topology. Dynamic topology nodes in GigaIO are nodes that may change in properties quite often. The terms “static” or “dynamic” should not be confused with the use of the terms in other contexts—their use in topology remains specific to GigaIO topology.

To specify dynamic nodes, they are tagged with :d[dynamic], for example (for the nodes node001..node004 that were set earlier):

Example
```
[head->partition*[base*]->fabrics*[GigaIO-1*]->topology]% get host
node001
node003
node004
[head->partition*[base*]->fabrics*[GigaIO-1*]->topology]% set host node001:d,node002,node003:d
[head->partition*[base*]->fabrics*[GigaIO-1*]->topology]% commit
[head->partition*[base*]->fabrics*[GigaIO-1*]->topology]% get host
node001:dynamic
node002
node003:dynamic
node004
```

The default specification—which is when there is no tag—means that the host is static, so that node004 remains as is.

- If, as in the preceding session, a switchport is not specified for the node, then dynamic tags are allocated by the set command in the order that the values for host are specified.

  The commit command is executed to complete the setting of the specified tag. In particular, the topology must not be applied—that is, the apply command must not be run.

- If a switchport is specified for the node, then it can set and committed separately. For example (continuing with the preceding session):

```
[head->partition*[base*]->fabrics*[GigaIO-1*]->topology]% set host s1p5 node002:d
[head->partition*[base*]->fabrics*[GigaIO-1*]->topology]% commit
[head->partition*[base*]->fabrics*[GigaIO-1*]->topology]% get host
node001:dynamic
node002:dynamic
node003:dynamic
node004
```

Setting Up Bindings
Once the topology is set up, the bindings can be set up.

Bindings—a simple conceptual example For example, using the single switch topology, 1S-2x4-1, the USPs and DSPs of the switch are meant to be allocated to nodes and resource boxes:
```
node001      node002
     /      \
    /        \__switch__
   /           /  \  
  dsp1    dsp2   dsp3    dsp4
io-01    io-02   io-03    io-04
```
The nodes and resource boxes may be connected physically with node001 connected to usp1, and node002 connected to usp2, and also dsp1 connected to io-01, and so on. This is represented by setting the parameters (set host node001..node002; set dsp io-01..io-04). The resulting connectivity can then be represented with:

```
node001 node002
  \  |
   usp1 usp2
  \ /  
   ----switch----
  / / \ / \ 
  dsp1 dsp2 dsp3 dsp4
  | | | |
io-01 io-02 io-03 io-04
```

How the resource is allocated to a node is known as the binding. Thus if io-01 and io-02 are bound to node001, and io-03 and io-04 are bound to node02, that is a particular binding configuration. The bindings are virtual, which means at the binding level the exact switch path is abstracted away. The configuration can then be represented with:

```
node001 node002
  \  |
   usp1 usp2
  \ /  
   ----switch----
  / / \ / \ 
  dsp1 dsp2 dsp3 dsp4
  | | | |
io-01 io-02 io-03 io-04
```

or simplifying it even further, as:

```
node001 node002
  \  |
   usp1 usp2
  \ /  
   ----switch----
  / / \ / \ 
  io-01 io-02 io-03 io-04
```

**Bindings—running the bind command, and interpreting the bindings shown in cmsh:** Continuing on with the 2S-4x4-3 topology session of earlier, the bindings of a fabric configuration can be seen with the `show` command:

```
[head->partition[base]->fabrics[GigaIO-1]->topology]% ..;bindings
[head->partition[base]->fabrics[GigaIO-1]->bindings]% show
Zone    USP      Links            DSPs
---------- ---------- ----------------- ------------
zone1     -        sis21k1..sis21k4    io-01..io-04
zone1    node001
zone1    node002
zone1    node003
zone1    node004
zone1    sis21k1
zone1    sis21k2
zone1    sis21k3
zone1    sis21k4
```

In the preceding, there are no bindings as yet, and all the links and resources of the zone1 pool are
listed. The line showing the pool itself is the entry without a USP, as indicated by the setting of “-” for the USP. A color-coded visual display for bindings can be seen with the `show -v|--visual` command (figure 3.26):

![Colorized Bindings: Nothing bound yet](image)

Continuing with the session, the `node001` switch port can be bound to the link `s1s2lk1` with:

```
bind node001 s1s2lk1
```

The `show` command then shows that `s1s2lk1` is now out of the pool, and is bound to `node001`.

```
show
Zone USP Links DSPs
-------- -------- -------- --------
zone1       node001  s1s2lk2..s1s2lk4  io-02..io-04
zone1       node001  s1s2lk1
zone1       node002
zone1       node003
zone1       node004
zone1       s1s2lk1
zone1       s1s2lk2
zone1       s1s2lk3
zone1       s1s2lk4
```

The `show -v` command for this configuration shows (figure 3.27):
The binding path that has just been created is displayed in the same color.

Continuing the session, the `io-01` switch port can be bound to the link `s1s2lk1` with:

```
% bind s1s2lk1 io-01
```

The `show` command then shows that the `io-01` resource box is out of the pool too, and is bound to `s1s2lk1`:

```
Zone  USP  Links                  DSPs
------- ------- ------------------- ------------
zone1   -      s1s2lk2..s1s2lk4  io-02..io-04
zone1 node001 s1s2lk1
zone1 node002
zone1 node003
zone1 node004
zone1 s1s2lk1
zone1 s1s2lk2
zone1 s1s2lk3
zone1 s1s2lk4
```

The `show -v` command for the state of the bindings then shows (figure 3.29):
The binding path is seen displayed in the same color, all the way from the node to the resource.

**Bindings—connection rules for bindings**: Bindings on a switch occur between entities that are links, resources, or nodes. The following connection rules must be followed for bindings for the two entities being bound, if both are in the same zone:

- Resources (DSP) can be bound to nodes on the same switch
- Links can be bound to a single node on the same switch
- Resources (DSP) can be bound to a single link on the same switch
- Links can be bound to links (these should not be circular in nature)

For example, binding connection attempts as indicated by the following diagrams fail:

- binding two nodes to one link:

  ![Diagram of two nodes connected to a single link]

  **fails**

- binding a node via a link back to the same node:

  ![Diagram of a node connected via a link to another node, then back to the first node]

  **fails**

- binding a node to a resource via two links:

  ![Diagram of a node connected to two resources via two links]
On the other hand, the bindings indicated by the following diagrams succeed:

- binding one node to two links, and having the two links then each connect to a resource:

  ![Diagram](image1)

  succeeds

- binding one node to a single link, and that link in turn connects to two resources:

  ![Diagram](image2)

  succeeds

**Bindings—applying and removing the bindings:**

- The *bindings* mode does not have a *commit* command, to help prevent the cluster administrator from automatically committing the change. The command used to carry out the binding is the *apply* command. This is because a binding change means that the PCI bus has changed, which is likely to upset the kernel of the host operating system of any attached system that is running, and cause severe issues. If the devices that are to be bound are on, then it is normally a good idea to power off the devices first, as suggested by a warning message:

  ```
  [head->partition[base]->fabrics[GigaIO-1]->bindings*]% apply
  Some devices are UP. power them off or use --force
  io-01, io-02, io-03, io-04, node001
  ```

  Using the -f|--force option is very unlikely to end well.

- To remove bindings, the *unbind* command can be used

  - per resource:
    ```
    [head->partition[base]->fabrics[GigaIO-1]->bindings*]% unbind io-01
    ```

  - over a resource range
    ```
    [head->partition[base]->fabrics[GigaIO-1]->bindings*]% unbind io-01..io-02
    ```

  - over a zone
    ```
    [head->partition[base]->fabrics[GigaIO-1]->bindings*]% unbind zone1
    ```

  - over a zone range
    ```
    [head->partition[base]->fabrics[GigaIO-1]->bindings*]% unbind zone1..zone2
    ```

  - over all resources:
    ```
    [head->partition[base]->fabrics[GigaIO-1]->bindings*]% unbind --all
    ```
### 3.15.6 Fabric Details With portmap

For regular switches, the `showport` command (section 3.8.4) can be used to find information about the switch ports used in a network.

For fabrics, the `portmap` command is used to get information about the fabric switch ports used in a fabric. Details on the links and PCI-e generations used can also be found:

```
[gigaio2] partition fabrics base
[gigaio2->partition[base]->fabrics] use myfabric
[gigaio2->partition[base]->fabrics[myfabric]] portmap
```

<table>
<thead>
<tr>
<th>Switch</th>
<th>Partition Ports</th>
<th>Ports</th>
<th>Speed</th>
<th>Negotiated speed</th>
<th>Status</th>
<th>Direction</th>
<th>LTSSM</th>
<th>Rate</th>
<th>Configured</th>
</tr>
</thead>
<tbody>
<tr>
<td>fswitch002</td>
<td>0 1-2 8 0</td>
<td>DOWN</td>
<td>USP</td>
<td>Detect</td>
<td>25.0GiT/s</td>
<td>16.0GiT/s</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>fswitch002</td>
<td>1 3-4 8 0</td>
<td>DOWN</td>
<td>USP</td>
<td>Detect</td>
<td>25.0GiT/s</td>
<td>16.0GiT/s</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>fswitch002</td>
<td>2 5-6 8 0</td>
<td>DOWN</td>
<td>USP</td>
<td>Detect</td>
<td>25.0GiT/s</td>
<td>16.0GiT/s</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>fswitch002</td>
<td>3 7-8 8 0</td>
<td>DOWN</td>
<td>USP</td>
<td>Detect</td>
<td>25.0GiT/s</td>
<td>16.0GiT/s</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>fswitch002</td>
<td>4 17-20 16 16</td>
<td>UP</td>
<td>USP</td>
<td>L0</td>
<td>16.0GiT/s</td>
<td>16.0GiT/s</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>fswitch002</td>
<td>5 21-24 16 16</td>
<td>UP</td>
<td>USP</td>
<td>L0</td>
<td>16.0GiT/s</td>
<td>16.0GiT/s</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>fswitch002</td>
<td>6 25 4 1</td>
<td>UP</td>
<td>USP</td>
<td>L0</td>
<td>5.0GiT/s</td>
<td>16.0GiT/s</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>fswitch001</td>
<td>0 1-2 8 8</td>
<td>UP</td>
<td>USP</td>
<td>L0</td>
<td>16.0GiT/s</td>
<td>16.0GiT/s</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>fswitch001</td>
<td>1 3-4 8 8</td>
<td>UP</td>
<td>USP</td>
<td>L0</td>
<td>16.0GiT/s</td>
<td>16.0GiT/s</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>fswitch001</td>
<td>2 5-6 8 8</td>
<td>UP</td>
<td>USP</td>
<td>L0</td>
<td>16.0GiT/s</td>
<td>16.0GiT/s</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>fswitch001</td>
<td>3 7-8 8 8</td>
<td>UP</td>
<td>USP</td>
<td>L0</td>
<td>16.0GiT/s</td>
<td>16.0GiT/s</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>fswitch001</td>
<td>4 17-20 16 16</td>
<td>UP</td>
<td>USP</td>
<td>L0</td>
<td>16.0GiT/s</td>
<td>16.0GiT/s</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>fswitch001</td>
<td>5 21-24 16 16</td>
<td>UP</td>
<td>USP</td>
<td>L0</td>
<td>16.0GiT/s</td>
<td>16.0GiT/s</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>fswitch001</td>
<td>6 25 4 1</td>
<td>UP</td>
<td>USP</td>
<td>L0</td>
<td>5.0GiT/s</td>
<td>16.0GiT/s</td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

```
[gigaio2->partition[base]->fabrics[myfabric]] portmap --leds
```

<table>
<thead>
<tr>
<th>Switch</th>
<th>Port</th>
<th>Partition</th>
<th>Neighbor</th>
<th>Cable ID</th>
<th>Present</th>
<th>Link</th>
<th>Direction</th>
<th>Maximum</th>
<th>Negotiated</th>
<th>Rate</th>
<th>Color</th>
</tr>
</thead>
<tbody>
<tr>
<td>fswitch002</td>
<td>1 0 0 0</td>
<td>no</td>
<td>USP</td>
<td>8 0</td>
<td>gen1</td>
<td>o----o</td>
<td>0 0</td>
<td>0 0</td>
<td>o----o</td>
<td></td>
<td></td>
</tr>
<tr>
<td>fswitch002</td>
<td>2 1 0</td>
<td>no</td>
<td>DSP</td>
<td>0 0</td>
<td>0 0</td>
<td>o----o</td>
<td>0 0</td>
<td>0 0</td>
<td>0 0</td>
<td></td>
<td></td>
</tr>
<tr>
<td>fswitch002</td>
<td>3 1 0</td>
<td>no</td>
<td>USP</td>
<td>8 0</td>
<td>gen1</td>
<td>o----o</td>
<td>0 0</td>
<td>0 0</td>
<td>o----o</td>
<td></td>
<td></td>
</tr>
<tr>
<td>fswitch002</td>
<td>4 1 0</td>
<td>no</td>
<td>DSP</td>
<td>0 0</td>
<td>0 0</td>
<td>o----o</td>
<td>0 0</td>
<td>0 0</td>
<td>o----o</td>
<td></td>
<td></td>
</tr>
<tr>
<td>fswitch002</td>
<td>5 2 0</td>
<td>no</td>
<td>USP</td>
<td>8 0</td>
<td>gen1</td>
<td>o----o</td>
<td>0 0</td>
<td>0 0</td>
<td>o----o</td>
<td></td>
<td></td>
</tr>
<tr>
<td>fswitch002</td>
<td>6 1 0</td>
<td>no</td>
<td>DSP</td>
<td>0 0</td>
<td>0 0</td>
<td>o----o</td>
<td>0 0</td>
<td>0 0</td>
<td>o----o</td>
<td></td>
<td></td>
</tr>
<tr>
<td>fswitch002</td>
<td>7 3 0</td>
<td>no</td>
<td>USP</td>
<td>8 0</td>
<td>gen1</td>
<td>o----o</td>
<td>0 0</td>
<td>0 0</td>
<td>o----o</td>
<td></td>
<td></td>
</tr>
<tr>
<td>fswitch002</td>
<td>8 1 0</td>
<td>no</td>
<td>DSP</td>
<td>0 0</td>
<td>0 0</td>
<td>o----o</td>
<td>0 0</td>
<td>0 0</td>
<td>o----o</td>
<td></td>
<td></td>
</tr>
<tr>
<td>fswitch002</td>
<td>9 1 0</td>
<td>APF0509011FBA</td>
<td>yes</td>
<td>USP</td>
<td>16 16</td>
<td>gen1</td>
<td>o----o</td>
<td>0 0</td>
<td>0 0</td>
<td>o----o</td>
<td></td>
</tr>
<tr>
<td>fswitch002</td>
<td>10 1 0</td>
<td>APF0509011F2P</td>
<td>yes</td>
<td>USP</td>
<td>8 8</td>
<td>gen1</td>
<td>o----o</td>
<td>0 0</td>
<td>0 0</td>
<td>o----o</td>
<td></td>
</tr>
<tr>
<td>fswitch002</td>
<td>11 1 0</td>
<td>APF0509011F81</td>
<td>yes</td>
<td>USP</td>
<td>8 8</td>
<td>gen1</td>
<td>o----o</td>
<td>0 0</td>
<td>0 0</td>
<td>o----o</td>
<td></td>
</tr>
<tr>
<td>fswitch002</td>
<td>12 1 0</td>
<td>APF0509011FBC</td>
<td>yes</td>
<td>USP</td>
<td>8 8</td>
<td>gen1</td>
<td>o----o</td>
<td>0 0</td>
<td>0 0</td>
<td>o----o</td>
<td></td>
</tr>
<tr>
<td>fswitch002</td>
<td>13 1 0</td>
<td>no</td>
<td>DSP</td>
<td>0 0</td>
<td>0 0</td>
<td>o----o</td>
<td>0 0</td>
<td>0 0</td>
<td>o----o</td>
<td></td>
<td></td>
</tr>
<tr>
<td>fswitch002</td>
<td>14 1 0</td>
<td>no</td>
<td>DSP</td>
<td>0 0</td>
<td>0 0</td>
<td>o----o</td>
<td>0 0</td>
<td>0 0</td>
<td>o----o</td>
<td></td>
<td></td>
</tr>
<tr>
<td>fswitch002</td>
<td>15 1 0</td>
<td>no</td>
<td>DSP</td>
<td>0 0</td>
<td>0 0</td>
<td>o----o</td>
<td>0 0</td>
<td>0 0</td>
<td>o----o</td>
<td></td>
<td></td>
</tr>
<tr>
<td>fswitch002</td>
<td>16 1 0</td>
<td>no</td>
<td>DSP</td>
<td>0 0</td>
<td>0 0</td>
<td>o----o</td>
<td>0 0</td>
<td>0 0</td>
<td>o----o</td>
<td></td>
<td></td>
</tr>
<tr>
<td>fswitch002</td>
<td>17 4 0</td>
<td>APF0399040431</td>
<td>yes</td>
<td>USP</td>
<td>16 16</td>
<td>gen1</td>
<td>o----o</td>
<td>0 0</td>
<td>0 0</td>
<td>o----o</td>
<td></td>
</tr>
<tr>
<td>fswitch002</td>
<td>18 1 1</td>
<td>APF0399040430</td>
<td>yes</td>
<td>USP</td>
<td>0 0</td>
<td>gen1</td>
<td>o----o</td>
<td>0 0</td>
<td>0 0</td>
<td>o----o</td>
<td></td>
</tr>
<tr>
<td>fswitch002</td>
<td>19 1 2</td>
<td>APF0399040435</td>
<td>yes</td>
<td>USP</td>
<td>0 0</td>
<td>gen1</td>
<td>o----o</td>
<td>0 0</td>
<td>0 0</td>
<td>o----o</td>
<td></td>
</tr>
<tr>
<td>fswitch002</td>
<td>20 1 3</td>
<td>APF0399040432</td>
<td>yes</td>
<td>USP</td>
<td>0 0</td>
<td>gen1</td>
<td>o----o</td>
<td>0 0</td>
<td>0 0</td>
<td>o----o</td>
<td></td>
</tr>
<tr>
<td>fswitch002</td>
<td>21 5 0</td>
<td>APF0399040439</td>
<td>yes</td>
<td>USP</td>
<td>16 16</td>
<td>gen1</td>
<td>o----o</td>
<td>0 0</td>
<td>0 0</td>
<td>o----o</td>
<td></td>
</tr>
<tr>
<td>fswitch002</td>
<td>22 1 1</td>
<td>APF0499047GKK</td>
<td>yes</td>
<td>USP</td>
<td>0 0</td>
<td>gen1</td>
<td>o----o</td>
<td>0 0</td>
<td>0 0</td>
<td>o----o</td>
<td></td>
</tr>
<tr>
<td>fswitch002</td>
<td>23 1 2</td>
<td>APF0399040436</td>
<td>yes</td>
<td>USP</td>
<td>0 0</td>
<td>gen1</td>
<td>o----o</td>
<td>0 0</td>
<td>0 0</td>
<td>o----o</td>
<td></td>
</tr>
<tr>
<td>fswitch002</td>
<td>24 1 3</td>
<td>APF0399040433</td>
<td>yes</td>
<td>USP</td>
<td>0 0</td>
<td>gen1</td>
<td>o----o</td>
<td>0 0</td>
<td>0 0</td>
<td>o----o</td>
<td></td>
</tr>
<tr>
<td>fswitch001</td>
<td>1 0 0</td>
<td>APF0509021FD3</td>
<td>yes</td>
<td>USP</td>
<td>8 8</td>
<td>gen1</td>
<td>o----o</td>
<td>0 0</td>
<td>0 0</td>
<td>o----o</td>
<td></td>
</tr>
</tbody>
</table>
The color column shows the led status at the FabreX port. In that column, an “o” represents an LED. The LED can be one of several colors, and it can be blinking. The meaning of the LED lights is covered by GigaIO documentation.

3.15.7 Monitoring Data Metrics For A Fabric
There are a number of fabric-related metrics that can be plotted to learn about how the fabric is being used over time.

```
gigaio2% partition use base
[gigaio2->partition[base]]% latestmetricdata | grep Fabric
FabricTopologies        Fabric 1  9.01s
FabricTopologyHostUsage Fabric 25.0%  9.01s
FabricTopologyResourceBoxUsage Fabric 33.3%  9.01s
```

```
gigaio2% network add fabrex
```
When the nodes come back up, a connectivity check can be carried out to verify node-to-node communications are working the FabreX interface, by using the `connectivity` command from within device mode:

```
[gigaio2->device]% connectivity --network fabrex --fabric myfabric
```

<table>
<thead>
<tr>
<th>Source</th>
<th>Destination</th>
<th>Result</th>
<th>ID</th>
<th>Latency</th>
</tr>
</thead>
<tbody>
<tr>
<td>node001</td>
<td>node001</td>
<td>Ok</td>
<td>0</td>
<td>0.1ms</td>
</tr>
<tr>
<td>node001</td>
<td>node002</td>
<td>Ok</td>
<td>0</td>
<td>0.2ms</td>
</tr>
<tr>
<td>node001</td>
<td>node003</td>
<td>Ok</td>
<td>0</td>
<td>0.2ms</td>
</tr>
<tr>
<td>node001</td>
<td>node004</td>
<td>Ok</td>
<td>0</td>
<td>0.2ms</td>
</tr>
<tr>
<td>node002</td>
<td>node001</td>
<td>Ok</td>
<td>0</td>
<td>0.2ms</td>
</tr>
<tr>
<td>node002</td>
<td>node002</td>
<td>Ok</td>
<td>0</td>
<td>0.1ms</td>
</tr>
<tr>
<td>node002</td>
<td>node003</td>
<td>Ok</td>
<td>0</td>
<td>0.2ms</td>
</tr>
<tr>
<td>node002</td>
<td>node004</td>
<td>Ok</td>
<td>0</td>
<td>0.2ms</td>
</tr>
<tr>
<td>node003</td>
<td>node001</td>
<td>Ok</td>
<td>0</td>
<td>0.2ms</td>
</tr>
<tr>
<td>node003</td>
<td>node002</td>
<td>Ok</td>
<td>0</td>
<td>0.2ms</td>
</tr>
<tr>
<td>node003</td>
<td>node003</td>
<td>Ok</td>
<td>0</td>
<td>0.1ms</td>
</tr>
<tr>
<td>node003</td>
<td>node004</td>
<td>Ok</td>
<td>0</td>
<td>0.2ms</td>
</tr>
<tr>
<td>node004</td>
<td>node001</td>
<td>Ok</td>
<td>0</td>
<td>0.3ms</td>
</tr>
<tr>
<td>node004</td>
<td>node002</td>
<td>Ok</td>
<td>0</td>
<td>0.2ms</td>
</tr>
<tr>
<td>node004</td>
<td>node003</td>
<td>Ok</td>
<td>0</td>
<td>0.2ms</td>
</tr>
<tr>
<td>node004</td>
<td>node004</td>
<td>Ok</td>
<td>0</td>
<td>0.1ms</td>
</tr>
</tbody>
</table>

The data values for latency are found using a non-optimized method. The values reported should therefore not be used as a latency benchmark for the interconnect.

### 3.15.9 Diagnosing FabreX Problems On The Nodes

If there are problems with connectivity over the FabreX network, then the easiest way to gain an insight into what is wrong is to log into a node and run the `fabrex-status.sh` utility:

```
root@node001:~# fabrex-status.sh
```

---

Host Details

- Hostname................: node001
- IP......................: 10.141.0.1
- IOMMU...................: Enabled
- Uptime..................: 5:06,1
- CPU usage................: 0.08, 0.08, 0.03 (1, 5, 15 min)
- Memory used.............: 1885 MB / 128764 MB
- Space on /..............:

---

Fabric Details
3.16 Cluster Configuration Without Execution By CMDaemon

3.16.1 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 Bright 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 Bright 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 3.16.2. 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 3.16.3.

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 3.16.4.

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 12.2–12.6. 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.

### 3.16.2 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 3.16.2), the following methods can be used:

- in Bright 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 3.16.2).

#### 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 3.16.4).

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@bright92 ~]# cmsh
   [bright92] % softwareimage
   [bright92->softwareimage]% clone default-image imagetwo
   [bright92->softwareimage*[imagetwo*]]% commit
   images/default-image -> /cm/images/imagetwo
   [bright92->softwareimage*[imagetwo*]]%
   ```
[bright92->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 12.2–12.6 of this chapter.

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:

   [bright92->softwareimage[imagetwo]]% category
   [bright92->category]% clone default categorytwo
   [bright92->category*[categorytwo*]]% set softwareimage imagetwo
   [bright92->category*[categorytwo*]]% commit
   [bright92->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):

   [bright92->category[categorytwo]]% device
   [bright92->device]% use node099
   [bright92->device[node099]]% set category categorytwo
   [bright92->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

   [bright92->device]% foreach -n node100..node200 (set category categorytwo)
   [bright92->device*] commit

5. Rebooting restarts the nodes that are assigned to the new category with the new image.

Modifying images via the cm/conf directory, for a category, node, or MAC address: The preceding 5-step method is understandable, but not very elegant. For NVIDIA Bright Cluster Manager 8.2 onwards, there is a more structured and efficient way to make some nodes function differently by image. It relies on adding 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.

### 3.16.3 Making All Nodes Function Differently From Normal Cluster Behavior With FrozenFile

Configuration changes carried out by Bright 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 Bright 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) 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.
3.16.4 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 3.16.2. 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/ifcf-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 Bright View clickpath:

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

3.16.5 Examples Of Configuring Nodes With Or Without CMDaemon

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

**Example**

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

A counterexample to this is:
Configuring a software for nodes without using Bright View or cmsh, 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 Bright 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 Bright View Or cmsh: Category Settings
When configuring nodes using Bright 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 Bright 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 3.16.2 to implement a functionality such as CUPS would load up CUPS as configured in an image, and would not handle it via CMDaemon. So, in section 3.16.2, 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 except to link nodes to their appropriate image via the associated category
Aspects of power management in NVIDIA Bright 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 CPU scaling governors for power-saving
- setting power-saving options in workload managers
- ensuring the passive head node can safely take over from the active head during failover (Chapter 18)
- 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.

Section 4.4 describes how CMDaemon can set CPU defaults to control some of the CPU-dependent power consumption.

The integration of power saving with workload management systems is covered in the chapter on Workload Management (section 7.9).

### 4.1 Configuring Power Parameters

Several methods exist to control power to devices:

- Power Distribution Unit (PDU) based power control
- IPMI-based power control (for node devices only)
- Custom power control
- HP iLO-based power control (for node devices only)
- Dell DRAC-based power control (for node devices only)
- Cisco UCS CIMC-based power control (for node devices only)
- Redfish-based power control (for node devices only)
4.1.1 PDU-Based Power Control

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. The device can then be turned on or off by changing the state of the PDU port.

To use PDU-based power control, the PDU itself must be a device in the cluster and be reachable over the network.

The PowerDistributionUnits menu option, accessible via the clickpath Devices[device] → Settings pane for each device plugged into the PDU, can then be used to configure the PDU ports that control the device. Figure 4.1 shows the menu option in the Settings window for a head node device, bright92, as accessed via the clickpath Devices → Head Nodes[bright92] → Settings.

Figure 4.1: Power Section In The Head Node Settings

If the menu option is clicked, then a subpane opens that allows each device plugged into the PDU to have PDU ports added and removed. For the APC brand of PDUs, the Power control property in the window of figure 4.1 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.

Since nodes may have multiple power feeds, there may be multiple PDU ports defined for a single device. The cluster management infrastructure takes care of operating all ports of a device in the correct order when a power operation is done on the device.

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.

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. Administrators wishing to use the cluster manager to record that a dumb PDU is assigned to a node can deal with it as follows:

- in Bright View the Notes field or the Userdefined1/Userdefined2 fields can be used.
4.1 Configuring Power Parameters

- in cmsh the equivalent is accessible when using the node from device mode, and running “set notes”, “set userdefined1”, or “set userdefined2”.

For PDUs that are manageable:

- In cmsh, power-related options can be accessed from device mode, after selecting a device:

  **Example**

  ```
  [bright92]% device use node001
  [bright92->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**

  ```
  [bright92->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:

<table>
<thead>
<tr>
<th>Power Status</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>ON</td>
<td>Power is on</td>
</tr>
<tr>
<td>OFF</td>
<td>Power is off</td>
</tr>
<tr>
<td>RESET</td>
<td>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.</td>
</tr>
<tr>
<td>FAILED</td>
<td>Power status script communication failure.</td>
</tr>
<tr>
<td>UNKNOWN</td>
<td>Power status script timeout</td>
</tr>
</tbody>
</table>

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 in figure 4.1 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:
Example

Configuring power parameters settings for a node using cmsh with APC:

```
[mycluster] % device use node001
[...device[physicalnode]] % set powerdistributionunits apc01:6 apc01:7 apc01:8
[...device[physicalnode]] % get powerdistributionunits
apc01:6 apc01:7 apc01:8
[...device[physicalnode]] % removefrom powerdistributionunits apc01:7
[...device[physicalnode]] % get powerdistributionunits
apc01:6 apc01:8
[...device[physicalnode]] % set powercontrol apc
[...device[physicalnode]] % get powercontrol
apc
[...device[physicalnode]] % 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 Bright 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 Bright 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:

```bash
#!/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:

```bash
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 over all nodes, the following steps are done:

1. The iLO interfaces of all nodes are set up like the IPMI interfaces as outlined in section 4.1.2, 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...

2. The ilo_power.pl custom power script must be configured on all nodes. This can be done with a cmsh script. For example, for all nodes in the default category:

   Example

   ```
   [mycluster]% device foreach -c default (set custompowerscript /cm/ilo\n   cal/apps/cmd/scripts/powerscripts/ilo_power.pl)
   [mycluster]% device foreach -c default (set powercontrol custom)
   [mycluster]% device commit
   ```

### 4.1.6 Dell drac-based Power Control

Dell drac configuration is covered on page 98.

### 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 Bright 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.
4.2 Power Operations

- **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 Bright View

In Bright 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 a head node `bright92` the button can be seen via the clickpath `Devices → Head Nodes[bright92] → Settings`, as illustrated in figure 4.1. Clicking on the button makes power operation buttons available figure 4.2.

![Figure 4.2: Actions Button, Accessing The Power Operations](image)

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 44). 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 856).

**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:

```
[bright92->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 13 on cluster monitoring has more on metrics and how they can be visualized.

4.4 CPU Scaling Governors

A cluster with CPU cores that run at a higher frequency consumes more power. A cluster administrator may therefore wish to implement schemes to control the CPU core frequencies, so that cluster power consumption is controlled.

4.4.1 The Linux Kernel And CPU Scaling Governors

In technology, a governor is the term used for a speed regulator. In computing, **CPU Scaling Governors** are power schemes that regulate what is commonly known as the CPU speed, or more precisely known
as the CPU core clock frequency. The Linux kernel uses the CPUFreq Governors interface to implement these power schemes. The following governor values are currently commonly available to the interface:

<table>
<thead>
<tr>
<th>Governor</th>
<th>Policy Description For CPU Frequency</th>
</tr>
</thead>
<tbody>
<tr>
<td>performance</td>
<td>set to the highest allowed</td>
</tr>
<tr>
<td>userspace</td>
<td>set to that determined by root</td>
</tr>
<tr>
<td>ondemand</td>
<td>set according to demand, aggressively</td>
</tr>
<tr>
<td>conservative</td>
<td>set according to demand, non-aggressively</td>
</tr>
<tr>
<td>powersave</td>
<td>set to the lowest allowed</td>
</tr>
</tbody>
</table>

### 4.4.2 The Governor List According To `sysinfo`

Not all governors may be available for a particular CPU. Also, new governors may appear as new hardware is released. However, the hardware-defined list of governors that the kernel does detect as being available to a node `<hostname>` can always be seen in the cluster manager:

- in Bright View from within the CPU information, within the System Information pane of the host. For example, the clickpath for the head node host `bright92` is Devices → Head Nodes[bright92] → System Information, and the information is displayed in the PROCESSORS subpane under the AVAILABLEGOVERNORS column.

- in `cmsh`, by using the `sysinfo <hostname>` command, in device mode

The list can be displayed under `cmsh` in a form such as:

**Example**

```
[bright92->device[bright92]]% sysinfo
System Information
------------------ --------------------------------------------------
... Number of Cores 12
Core 0 Six-Core AMD ... performance (ondemand, userspace)
Core 1 Six-Core AMD ... performance (ondemand, userspace)
... 
```

The governor in active operation for the cores is displayed outside the parentheses—in the example it is `performance`. The other two available, but inactive ones, are enclosed by parentheses—in the example they are `ondemand` and `userspace`. This list is essentially a hardware-defined list because its members depend only on the chip used. Which one of the members is active is however decided by a software setting for the governor value.

### 4.4.3 Setting The Governor

CPU scaling—Cool ‘n’ Quiet (AMD) or SpeedStep (Intel)—must be enabled in the BIOS for the governor to operate. Values for the governor may then be set in the BIOS. If a governor value is set in the BIOS, then the operating system cannot override it. If no value is set in the BIOS, then the value is set by the operating system.

On a running system, the cluster administrator can set the CPU governor value by using the CM-Daemon front ends as follows:

- In Bright View:
– if the Node Categories resource is chosen, and a particular category is selected, then a Settings windows option for that category allows setting the Scaling governor to:

* a single value.
* multiple values. These can be set as a comma-separated list. This should not be confused with the hardware-defined list (section 4.4.2).

– if the Nodes or Head Nodes resources are chosen, and a node selected from there, then its Settings window option allows its Scaling governor value to be set only to a single value.

- In cmsh:

– if category mode, is used, and the category is chosen, then the scalinggovernor can be set to:

* a single value.
* multiple values. These can be set as a comma-separated list

– if device mode is chosen, and a head or regular node selected from there, then its scalinggovernor value can be set only to a single value.

If a list is set for scaling governors in the cluster manager, at category level, then the first value in the list is applicable to all the nodes in that category. If that fails for any of the nodes, for example if a node does not support this value, then the next value is attempted, and so on, until the node is assigned a governor value. This allows configuration according to preferred order to take place at category level.

As usual, configuration of a value at node level overrides the category value. However, in contrast with the setting that is possible at the category level, only a single value is possible at node level. This is because, if a specific node comes to the attention of the administrator for a custom governor value, then that node can be assigned the value decided by the administrator.

Example

[bright92->]% device use node001
[bright92->device[node001]]% sysinfo
System Information
--------------------------------- --------------------------------------------------
... Number of Cores 12
Core 0 Six-Core AMD ... performance (ondemand, userspace)
Core 1 Six-Core AMD ... performance (ondemand, userspace)
... Core 11 Six-Core AMD ... performance (ondemand, userspace)

[bright92->device[node001]]% category use default
[bright92->category[default]]% set scalinggovernor ondemand,userspace,performance
[bright92->category[default]]% device use node001
[bright92->device[node001]]% sysinfo
System Information
--------------------------------- --------------------------------------------------
... Number of Cores 12
Core 0 Six-Core AMD ... ondemand (userspace, performance)
Core 1 Six-Core AMD ... ondemand (userspace, performance)
... Core 11 Six-Core AMD ... ondemand (userspace, performance)
4.5 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.
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 Bright 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 201) 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**

  ```bash
  [root@bright92 ~]# cmsh
  [bright92]% device use node001
  [bright92->device[node001]]% set pxelabel MEMTEST ; commit
  ```

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

  **Example**

  ```bash
  [root@bright92 ~]# cmsh
  [bright92]% device
  [bright92->device]% foreach -c default (set pxelabel MEMTEST)
  [bright92->device*]% commit
  ```

  The value of pxelabel can be cleared with:

  **Example**

  ```bash
  [root@bright92 ~]# cmsh -c "device; foreach -c default (clear pxelabel); commit"
  ```

  In Bright View, the PXE label can be set from the Settings window for a node (figure 5.2).

- 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 Bright 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/`. 

---

Figure 5.2: Bright View PXE Label option
5.1 Before The Kernel Loads

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 12.

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 78) can be set from cmsh or Bright 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 Bright 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 \texttt{BOOTIF}), must not be an interface that is already configured for that node in CMDaemon. For example, if \texttt{BOOTIF} is the device \texttt{ib0}, then \texttt{ib0} must not already be configured in CMDaemon. Either \texttt{BOOTIF} or the \texttt{ib0} configuration should be changed so that node installation can succeed. It is recommended to set \texttt{BOOTIF} to \texttt{eth0} if the \texttt{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 OmniPath, 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 \texttt{PXE LABEL} to \texttt{localdrive} (page 191).
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

```
[bright92->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

```
[bright92->device[node001]]% get bootloaderprotocol
HTTP (default-centos7-x86_64)
[bright92->device[node001]]% set bootloaderprotocol<TAB><TAB>http https tftp
[bright92->device[node001]]% set bootloaderprotocol tftp
[bright92->device[node001]]% commit
[bright92->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:

<table>
<thead>
<tr>
<th>Property</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>allImages</td>
<td>The following values decide what images the provisioning node provides:</td>
</tr>
<tr>
<td></td>
<td>• onlocaldisk: all images on the local disk, regardless of any other parameters set</td>
</tr>
<tr>
<td></td>
<td>• onsharedstorage: all images on the shared storage, regardless of any other parameters set</td>
</tr>
<tr>
<td></td>
<td>• no (the default): only images listed in the localimages or sharedimages parameters, described next</td>
</tr>
<tr>
<td>localimages</td>
<td>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”.</td>
</tr>
<tr>
<td>sharedimages</td>
<td>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”</td>
</tr>
<tr>
<td>Provisioning slots</td>
<td>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.</td>
</tr>
<tr>
<td>nodegroups</td>
<td>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.</td>
</tr>
</tbody>
</table>

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

```
[bright92]% category add misc
[bright92->category[misc*]]% roles
[bright92->category[misc*]->roles% assign provisioning
[bright92->roles[provisioning*]]% set allimages no
[bright92->roles[provisioning*]]% set localimages default-image
[bright92->roles[provisioning*]]% set provisioningslots 20
[bright92->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
[bright92->category[misc*]->roles[provisioning*]]% commit
[bright92->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

```
[bright92]% device use node001
[bright92->device[node001]]% roles
[bright92->device[node001]->roles% assign provisioning
[bright92->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.
5.2 Provisioning Nodes

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 Bright View

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

A misc category can be added via the clickpath
Grouping → Categories → Add → Settings → <name>

The node category should be given a name misc (figure 5.3), and saved:

![Figure 5.3: Bright View: Adding A misc Category](figure5.3)

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):
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):
5.2 Provisioning Nodes

The Provisioning slots setting decides, for example, how many images can be supplied simultaneously from the provisioning node, while the Images and All images settings decide what images the provisioning node supplies. As before, the setting can be saved using the Save button of figure 5.3.

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 Bright 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 (Bright 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, it is scheduled to run every midnight by default.

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**

```
[bright92] % softwareimage updateprovisioners
Provisioning nodes will be updated in the background.

Sun Dec 12 13:45:09 2010 bright92: Starting update of software image(s)
provisioning node(s). (user initiated).
[bright92] % softwareimage updateprovisioners [bright92]%
Sun Dec 12 13:45:41 2010 bright92: Updating image default-image on prov
isioning node node001.
[bright92] %
Sun Dec 12 13:46:00 2010 bright92: Updating image default-image on prov
isioning node node001 completed.
Sun Dec 12 13:46:00 2010 bright92: Provisioning node node001 was updated
Sun Dec 12 13:46:00 2010 bright92: 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**

```
[bright92->softwareimage] % drain -n master
Nodes drained
[bright92->softwareimage] % provisioningstatus
Provisioning subsystem status
Pending request: node001, node002
Provisioning node status:
+ bright92
  Slots: 1 / 10
  State: draining
  Active nodes: node003
  Up to date images: default-image
[bright92->softwareimage] % provisioningstatus
Provisioning subsystem status
Pending request: node001, node002
Provisioning node status:
+ bright92
  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**

```
```
Provisioning Node Update Safeguards And \texttt{provisioningnodeautoupdatetimeout}

The \texttt{updateprovisioners} command is subject to safeguards that prevent it running too frequently. The minimum period between provisioning updates can be adjusted with the parameter \texttt{provisioningnodeautoupdatetimeout}, 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 timeout is set to zero (\emph{false}).

The parameter can be accessed and set within \texttt{cmsh} from partition mode:

\textbf{Example}

\begin{verbatim}
[root@bright92 ]# cmsh
[bright92]% partition use base
[bright92->partition[base]]% get provisioningnodeautoupdatetimeout
[bright92->partition[base]]% 300
[bright92->partition[base]]% set provisioningnodeautoupdatetimeout 0
[bright92->partition[base]]% commit
\end{verbatim}

Within Bright View the parameter is accessible via the clickpath: Cluster $\rightarrow$ Partition[base] $\rightarrow$ Provisioning Node 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.

\textbf{Synchronization Of Fspart Subdirectories To Provisioning Nodes}

In the cluster manager, an \texttt{fspart} is a subdirectory, and it is a filesystem part that can be synced during provisioning.

The fsparts can be listed with:

\textbf{Example}

\begin{verbatim}
[root@bright92 ]# cmsh
[bright92]% fspart
[bright92->fspart]% list

<table>
<thead>
<tr>
<th>Path (key)</th>
<th>Type</th>
<th>Image</th>
</tr>
</thead>
<tbody>
<tr>
<td>/cm/images/default-image</td>
<td>image</td>
<td>default-image</td>
</tr>
<tr>
<td>/cm/images/default-image/boot</td>
<td>boot</td>
<td>default-image:boot</td>
</tr>
<tr>
<td>/cm/node-installer</td>
<td>node-installer</td>
<td></td>
</tr>
<tr>
<td>/cm/shared</td>
<td>cm-shared</td>
<td></td>
</tr>
<tr>
<td>/tftpboot</td>
<td>tftpboot</td>
<td></td>
</tr>
<tr>
<td>/var/spool/cmd/monitoring</td>
<td>monitoring</td>
<td></td>
</tr>
</tbody>
</table>
\end{verbatim}

The \texttt{updateprovisioners} command (page 201) is used to update image fsparts to all nodes with a provisioning role.

\textbf{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 \texttt{--all} option:
Example

[bright92->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.

[bright92->fspart]% info

<table>
<thead>
<tr>
<th>Path</th>
<th>Architecture</th>
<th>OS</th>
<th>Notify watchers</th>
</tr>
</thead>
<tbody>
<tr>
<td>/cm/images/default-image</td>
<td>x86_64</td>
<td>rhel7</td>
<td>0</td>
</tr>
<tr>
<td>/cm/images/default-image/boot</td>
<td>-</td>
<td>-</td>
<td>0</td>
</tr>
<tr>
<td>/cm/node-installer</td>
<td>x86_64</td>
<td>rhel7</td>
<td>0</td>
</tr>
<tr>
<td>/cm/shared</td>
<td>x86_64</td>
<td>rhel7</td>
<td>0</td>
</tr>
<tr>
<td>/tftpboot</td>
<td>-</td>
<td>-</td>
<td>0</td>
</tr>
<tr>
<td>/var/spool/cmd/monitoring</td>
<td>-</td>
<td>-</td>
<td>0</td>
</tr>
</tbody>
</table>

[bright92->fspart]% info -s

<table>
<thead>
<tr>
<th>Path</th>
<th>Architecture</th>
<th>OS</th>
<th>Notify watchers</th>
<th>Size</th>
</tr>
</thead>
<tbody>
<tr>
<td>/cm/images/default-image</td>
<td>x86_64</td>
<td>rhel7</td>
<td>0</td>
<td>4.2 GiB</td>
</tr>
<tr>
<td>/cm/images/default-image/boot</td>
<td>-</td>
<td>-</td>
<td>0</td>
<td>179 MiB</td>
</tr>
<tr>
<td>/cm/node-installer</td>
<td>x86_64</td>
<td>rhel7</td>
<td>0</td>
<td>2.45 GiB</td>
</tr>
<tr>
<td>/cm/shared</td>
<td>x86_64</td>
<td>rhel7</td>
<td>0</td>
<td>2.49 GiB</td>
</tr>
<tr>
<td>/tftpboot</td>
<td>-</td>
<td>-</td>
<td>0</td>
<td>3.3 MiB</td>
</tr>
<tr>
<td>/var/spool/cmd/monitoring</td>
<td>-</td>
<td>-</td>
<td>0</td>
<td>1.02 GiB</td>
</tr>
</tbody>
</table>

The locked, lock, and unlock commands:

- The `locked` command lists fsparts that are prevented from syncing.

Example

[bright92->fspart]% locked
No locked fsparts

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

Example

[bright92->fspart]% lock /var/spool/cmd/monitoring
[bright92->fspart]% locked /var/spool/cmd/monitoring

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

Example

[bright92->fspart]% unlock /var/spool/cmd/monitoring
[bright92->fspart]% locked
No locked fsparts
Access to `excludelistsnippets`: The properties of `excludelistsnippets` for a specific fs part can be accessed from the `excludelistsnippets` submode:

Example

```
[bright92->fspart]% excludelistsnippets /tftpboot

[bright92->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 yes no no

[bright92->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

[bright92->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 12 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 Bright 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 Bright View**

In Bright View the selection of kernel modules for a particular image, is carried out through the Software images resource, and then choosing the Kernel modules menu option of that image. For example, for the image default-image, clickpath Provisioning→Software images[default-image]→Edit→Settings→Kernel modules, leading to figure 5.6:
5.3 The Kernel Image, Ramdisk And Kernel Modules

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 Bright 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 12.3). 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 12.3.4.

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:

```bash
[root@bright92 ~]# { service rdma stop; lsmod | cut -f1 -d" "; }>/tmp/a
[root@bright92 ~]# { 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@bright92 ~]# diff /tmp/a /tmp/b
1,3c1
< Unloading OpenIB kernel modules:
< Failed to unload ib_core
<                        [FAILED]
---
> Loading OpenIB kernel modules:
> [ OK ]
4a3,14
> ibipoib
> 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 `ibipoib`, `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 onwards, 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@bright92 ~]# cmsh
[bright92]# softwareimage use default-image
[bright92->softwareimage[default-image]% kernelmodules
  [bright92...age[default-image]->kernelmodules]% add mlx4_ib
  [bright92...age*[default-image*]->kernelmodules*[mlx4_ib*]]% add ib_ipoib
  [bright92...age*[default-image*]->kernelmodules*[ib_ipoib*]]% add ib_umad
  [bright92...age*[default-image*]->kernelmodules*[ib_umad*]]% commit
[bright92->softwareimage[default-image]->kernelmodules[ib_umad]]%
Tue May 24 03:45:35 2011 bright92: Initial ramdisk for image default-image was regenerated successfully.
```

If the modules are put in another image instead of default-image, then the default image that nodes boot from should be set to the new image (section 3.16.2).

### 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@bright92 ~]# cmsh
[bright92]# device
[bright92->device]% foreach -n node001..node150 (set provisioninginterface ib0)
[bright92->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@bright92 ~]# cmsh
[bright92]# softwareimage use default-image
[bright92->softwareimage[default-image]% kernelmodules
  [bright92...age[default-image]->kernelmodules]% add hfi1
  [bright92...age*[default-image*]->kernelmodules*[hfi1*]]% add ib_ipoib
  [bright92...age*[default-image*]->kernelmodules*[ib_ipoib*]]% commit
[bright92->softwareimage[default-image]->kernelmodules[ib_ipoib]]%
Thu Jul 4 15:03:35 2019 bright92: Initial ramdisk for image default-image 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@bright92 ~]# cmsh
[bright92] % device
[bright92->device] % foreach -n node001..node150 (set provisioninginterface ib0)
[bright92->device*] % commit
```

5.3.5 VLAN Provisioning
Nodes can be configured for provisioning over a VLAN interface, starting in NVIDIA Bright 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 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 12.3.4). An example where the VLANID is appended to some existing parameters could be:

Example

```
[root@bright92 ~]# cmsh
[bright92] % softwareimage
[bright92->softwareimage] % use default-image
[bright92->softwareimage[default-image]] % append kernelparameters " VLANID=89"
[bright92->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 12.4.4) 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.3) on the node it is installing to, so that they differ from the otherwise-expected pre-init stage Linux system. Such modifications can be prevented by a `frozenFilesPerNode` or `frozenFilesPerCategory` directive, as documented within the `node-installer.conf` file, and explained in greater detail on page 857.

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 9 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).

![Certificate Request]

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@bright92 ~]# cmsh
[bright92]# partition
[bright92->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 auto-signing 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@bright92 ~]# cmsh
[bright92]# network use internalnet
[bright92->network[internalnet]]% set allowautosign automatic
```

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:

```
[bright92->cert]% listrequests
Request ID Client type Session ID Autosign Name
-------------- -------------- -------------- ------- -------------------------------
6 installer 42949672986 No fa-16-3e-22-cd-13
```

```
[bright92->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
5.4 Node-Installer

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.
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 5.11: Scenarios: Unknown Node, No Ethernet Port Detected
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.
A truth table summarizing the scenarios is helpful:

<table>
<thead>
<tr>
<th>Scenario</th>
<th>Node known?</th>
<th>Switch port detected?</th>
<th>Switch port configuration found?</th>
<th>Switch port configuration conflicts with node configuration?</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>No</td>
<td>Yes</td>
<td>Yes</td>
<td>No</td>
</tr>
<tr>
<td>2</td>
<td>No</td>
<td>Yes</td>
<td>No</td>
<td>No</td>
</tr>
<tr>
<td>3</td>
<td>No</td>
<td>No</td>
<td>No</td>
<td>No</td>
</tr>
<tr>
<td>4</td>
<td>Yes</td>
<td>Yes</td>
<td>Yes</td>
<td>No</td>
</tr>
<tr>
<td>5</td>
<td>Yes</td>
<td>Yes</td>
<td>Yes</td>
<td>Yes (configurations differ)</td>
</tr>
<tr>
<td>6</td>
<td>Yes</td>
<td>No</td>
<td>Yes</td>
<td>Yes (port expected by MAC configuration not found)</td>
</tr>
<tr>
<td>7</td>
<td>Yes</td>
<td>Yes</td>
<td>No</td>
<td>No (port not expected by MAC configuration)</td>
</tr>
</tbody>
</table>

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)
5.4 Node-Installer

- 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**

  ```
  [bright92->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**

  ```
  [bright92->device]% showport 00:0C:29:01:0F:F8
  switch01:8
  [bright92->device]% set node003 switchports switch01:8
  [bright92->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 is to use the `--clone` option of the `foreach` command (section 2.5.5).

  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:
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

[bright92->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,
ode022 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

[bright92->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

[bright92->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:

<table>
<thead>
<tr>
<th>before</th>
<th>after</th>
</tr>
</thead>
<tbody>
<tr>
<td>node020</td>
<td>node020</td>
</tr>
<tr>
<td>node021</td>
<td>node021</td>
</tr>
<tr>
<td>node022</td>
<td>node022</td>
</tr>
<tr>
<td>node023</td>
<td>...A</td>
</tr>
<tr>
<td>node024</td>
<td>...B</td>
</tr>
<tr>
<td>node025</td>
<td>...C</td>
</tr>
</tbody>
</table>

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

[bright92->device] % newnodes -s -o 1 node024..node025
then the rack layout looks like:

<table>
<thead>
<tr>
<th>before</th>
<th>after</th>
</tr>
</thead>
<tbody>
<tr>
<td>node020</td>
<td>node020</td>
</tr>
<tr>
<td>node021</td>
<td>node021</td>
</tr>
<tr>
<td>node022</td>
<td>node022</td>
</tr>
<tr>
<td>unassigned</td>
<td>...A</td>
</tr>
<tr>
<td>node024</td>
<td>...B</td>
</tr>
<tr>
<td>node025</td>
<td>...C</td>
</tr>
</tbody>
</table>

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**

```
[bright92->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:

<table>
<thead>
<tr>
<th>before</th>
<th>after</th>
<th>MAC</th>
</tr>
</thead>
<tbody>
<tr>
<td>node020</td>
<td>node020</td>
<td></td>
</tr>
<tr>
<td>node021</td>
<td>node021</td>
<td></td>
</tr>
<tr>
<td>node022</td>
<td>node022</td>
<td></td>
</tr>
<tr>
<td>node024</td>
<td>...A</td>
<td></td>
</tr>
<tr>
<td>node025</td>
<td>...B</td>
<td></td>
</tr>
<tr>
<td>node026</td>
<td>...C</td>
<td></td>
</tr>
</tbody>
</table>

`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 Bright 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**

```
[bright92->device]% foreach -n node020..node029 (set rack rack02)
[bright92->device*]% commit
[bright92->device]% foreach -n node030..node039 (set rack rack03)
[bright92->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**

```
[bright92->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**
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:

  ```bash
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 Bright View equivalent to the newnodes command of cmsh. Like newnodes, the resource lists the MAC address of any unassigned node that the head node detects, and shows the associated detected switch port for the node. Also, like newnodes, it 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
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 nodes, 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 by running cmsh's foreach loop command on a node with a --clone option (section 2.5.5).

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 auto, which means 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 available numbers, say node328-node337.

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 config-
uration (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 227) 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 229) 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

²For example, **imageupdate** (section 5.6.2), which is run by CMDaemon, ignores these settings, which is as expected. This means that, for example, if **imageupdate** is run with **NOSYNC** set, then the head node image is still synchronized over as usual to the regular node while the node is up. It is only during node boot, during the installer stage, that setting **NOSYNC** prevents synchronization.
• If the install-mode is set to FULL, then the node-installer re-partitions, creates new filesystems and synchronizes a full image onto the local drive according a partition layout. This process wipes out all pre-boot drive content.

A partition layout (Appendix D) includes defined values for the partitions, sizes, and filesystem types for the nodes being installed. An example of a partition layout is the default partition layout (Appendix D.3).

• If the install-mode is set to AUTO, then the node-installer checks the partition layout of the local drive against the node’s stored configuration. If these do not match because, for example, the node is new, or if they are corrupted, then the node-installer recreates the partitions and filesystems by carrying out a FULL install. If however the drive partitions and filesystems are healthy, the node-installer only does an incremental software image synchronization. Synchronization tends to be quick because the software image and the local drive usually do not differ much.

Synchronization also removes any extra local files that do not exist on the image, for the files and directories considered. Section 5.4.7 gives details on how it is decided what files and directories are considered.

• If the install-mode is set to MAIN, then the node-installer does not carry out a disk check, and goes on to maintenance mode, allowing manual investigation of specific problems. The local drive is untouched.

• If the install-mode is set to NOSYNC, and the partition layout check matches the stored XML configuration, then the node-installer skips synchronizing the image to the node, so that contents on the local drive persist from the previous boot. An exception to this is the node certificate and key, that is the files /cm/local/apps/cmd/etc/cert.{pem|key}. These are updated from the head node if missing.

If however the partition layout does not match the stored configuration, a FULL image sync is triggered. Thus, for example, a burn session (Chapter 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 229).

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

[bright92] % device foreach -n node001..node999 (set nextinstallmode nosync)
[bright92] % 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
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.
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 Bright 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 Bright View with a clickpath such as:
5.4 Node-Installer

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:

![Install-mode Setting Option During Node-Installer Run](image)

**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 Bright 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@bright92 ~]# cmsh -c "device use node001; set datanode yes; commit"
```

The property can also be set at a category or other levels.

**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 Bright 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

[bright92->device]% installerinteractions -v -n node001 --confirm
Hostname Action
--------- ---------------------------------------------
node001 Requesting FULL Install (partition mismatch)
[bright92->device]%
...07:57:36 [notice] bright92: node001 [ INSTALLER_CALLINGINIT ]...
[bright92->device]%
...07:58:20 [notice] bright92: node001 [ UP ]

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 42).

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:

[bright92->device[node001]]% append blockdevicesclearedonnexboot /dev/sda /dev/sdb ; commit

The value of blockdevicesclearedonnexboot is automatically cleared after the node is rebooted.
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 3.16.4). 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:

<table>
<thead>
<tr>
<th>Option</th>
</tr>
</thead>
<tbody>
<tr>
<td>64kB</td>
</tr>
<tr>
<td>128kB</td>
</tr>
<tr>
<td>256kB</td>
</tr>
<tr>
<td>512kB</td>
</tr>
<tr>
<td>1024kB</td>
</tr>
</tbody>
</table>

- stripe size:
232 Node Provisioning

- **cache policy:**
  - Cached
  - Direct

<table>
<thead>
<tr>
<th>Option</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>NORA</td>
<td>No Read Ahead</td>
</tr>
<tr>
<td>RA</td>
<td>Read Ahead</td>
</tr>
<tr>
<td>ADRA</td>
<td>Adaptive Read</td>
</tr>
</tbody>
</table>

- **write policy:**
  - 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@bright92 ~]# cmsh
[bright92] softwareimage list
Name (key) Path Kernel version Nodes
------------------------- ------------------------------------------------- ---------------------------- --------
default-image /cm/images/default-image 3.10.0-1062.12.1.el7.x86_64 3
[bright92] softwareimage
[bright92->softwareimage] lock default-image
[bright92->softwareimage] islocked
Name Locked
------------------------- --------
default-image yes
```
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:

```bash
cmsh -c "category use default; get excludelistfullinstall"
```

Similarly, to set the list:

```bash
cmsh -c "category use default; set excludelistfullinstall; commit"
```
where a text-editor opens up to allow changes to be made to the list. In Bright 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 exclude list full install and exclude list sync install properties decide how a node synchronizes to an image during boot. For a node that is already fully up, the related exclude list update 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

```
[bright92->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 18), 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 18.1).

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:

- rsyncaemon, which sends the data unencrypted
- rsynccsh, 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 rsynccsh is only suggested if the users on the network cannot be trusted. By default, provisioningtransport is set to rsyncaemon. If high availability (Chapter 18) is set up with the head nodes exposed to the outside world on the external network, the administrator should consider setting up rsynccsh for the head nodes.

The rsynccsh 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 12.6.2), the keys must be copied over from /root/.ssh/ on the existing nodes.
5.4 Node-Installer

**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:

```
bright92:~ # 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 bright92:
  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 Bright 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*

```
[bright92->device]# synclog node001
Tue, 11 Jan 2011 13:27:17 CET - Starting rsync daemon based provisioning. Mode is SYNC.

sending incremental file list
./
... deleted 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

```

**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 transferred, 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 Transferred 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:

```
bright92:~ # 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:

```
bright92:~ # 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 Bright 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 Bright View or `cmsh` (section 3.10.2).

### 5.4.10 Installing GRUB Bootloader

By default, a node-installer boots from the software image on the head node via the network.

Optionally, the node-installer installs a boot record on the local drive if the `installbootrecord` property of the node configuration or node category is set to `on`, so that the next boot can be from the local drive.

For a 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 occurs, despite whatever value `installbootrecord` has.
2. the GRUB bootloader with a boot record must be installed in the MBR of the local drive, overwriting the default iPXE boot record.

If the administrator is not using the default software image, but is using a custom software image (section 12.6.1), and if the image is based on a running node filesystem 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 12.6.2) 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.

With a working custom software image, to set the GRUB bootloader in Bright View, the Install boot record option must be enabled and saved in the node configuration or in the node category. The cmsh equivalents are commands like:

```
cmsh -c "device use node001; set installbootrecord yes; commit"
```

or

```
cmsh -c "category use default; set installbootrecord yes; 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 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 the boot record has been installed as in the previous two items, 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 Bright 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@bright92 ~]# 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**

```
[bright92%] device interfaces node001
[bright92%] device[node001]->interfaces% list
Type Network device name IP Network
-------------- --------------- -----------
```
physical BOOTIF [prov] 10.141.0.1 internalnet
[bright92->device[node001]->interfaces]% set bootif networkdevicename eth0
[bright92->device[node001]->interfaces]% commit
[bright92->device[node001]->interfaces]% list

<table>
<thead>
<tr>
<th>Type</th>
<th>Network device name</th>
<th>IP</th>
<th>Network</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>physical eth0 [prov]</td>
<td>10.141.0.1</td>
<td>internalnet</td>
</tr>
</tbody>
</table>

Tue Mar 31 13:46:44 2020 [notice] bright92: 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 9 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 3.16.4) 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 12.7) 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 567.

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 Bright View
In the node icons used by Bright View:

- Nodes in the UP state are indicated by an up-arrow.
  - If all health checks (section 13.2.4) for the node are successful, the up-arrow is green.
  - If there is a health check that fails or if the node requires a reboot, the up-arrow is red.

- Nodes in the DOWN state are indicated by a blue down-arrow.

- There are some other states, including:
  - Nodes in a CLOSED state are indicated by an X
  - Nodes in a DOWN state that are installing are indicated by a underscored down-arrow icon: ↓
5.5.2 Node States Shown In cmsh

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

Example

[bright92->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 32) command:

Example

[bright92->device]% list -f "hostname:10, status:48"
hostname | status
---------- ------------------------------------------------
apc01 [ UP ]
bright92 [ UP ]
devhp [ UP ]
node001 [ UP ] restart-required, health check failed
node002 [ DOWN ] (hostname changed) restart-required

The reason for a red icon as shown in section 5.5.1 can be found within the parentheses. In this example it is (hostname changed).

5.5.3 Node States Indicating Regular Start Up

During a successful boot process the node goes through the following states:

- **BOOTING**. This is the state while the kernel and initrd are being downloaded by the node during network booting.

  To allow the BOOTING state to be detected for a node:

  - **BOOTIF** must be defined as an interface, or
  - if there is no interface with the value **BOOTIF**, but a particular network device, such as eth0, is the boot interface, then a special revision tag of bootif can be used for no more than one interface:

    Example

    [bright92->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 Bright 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:
5.5 Node States

(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 Bright 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 Bright View via the clickpath

Devices->Nodes->Edit Monitor 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.
  * **--failbeforedown <count>**: Specifies the number of failed pings before a device is marked as down (default is 1).

- **drain** and **undrain** (Appendix G.4.1)

- For nodes that have power control:
  * `power -f on`
  * `power -f off`
  * `power -f reset`

  In Bright 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
5.6 Updating Running Nodes

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 Bright 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 Bright 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 Synchronize image (Bright View), and involves further exclude lists excludelistgrab or excludelistgrabnew. The grabimage command and Synchronize image option are covered in detail in section 12.5.2.
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 `excludelistupdate` list that is being discussed here, in contrast with the `excludelistsyncinstall` and `excludelistfullinstall` lists which are discussed in section 5.4.7, and also in contrast with the `excludelistgrab` and `excludelistgrabnew` lists of section 12.5.2.

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` and `excludelistgrabnew` lists of section 12.5.2 are about a *grab* from the running node to the image. The `excludelistgrab` list is here intended for synchronizing the existing image with the running node, and is thus analogous to the `excludelistsyncinstall` exclusion list, while the `excludelistgrabnew` list is intended for copying a full new image from the running node, and is thus analogous to the `excludelistfullinstall` list.

The following table summarizes this:

<table>
<thead>
<tr>
<th>During:</th>
<th>Exclude list used is:</th>
<th>Copy intention:</th>
</tr>
</thead>
<tbody>
<tr>
<td>update</td>
<td><code>excludelistupdate</code></td>
<td>sync, image to running node</td>
</tr>
<tr>
<td>install</td>
<td><code>excludelistfullinstall</code> <code>excludelistsyncinstall</code></td>
<td>full, image to starting node sync, image to starting node</td>
</tr>
<tr>
<td>grab</td>
<td><code>excludelistgrabnew</code> <code>excludelistgrab</code></td>
<td>full, running node to image sync, running node to image</td>
</tr>
</tbody>
</table>

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:  

<table>
<thead>
<tr>
<th>Exclusive List Used Is:</th>
<th>Copy Intention:</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>update</code></td>
<td><code>excludelistupdate</code></td>
</tr>
<tr>
<td><code>install</code></td>
<td><code>excludelistfullinstall</code></td>
</tr>
<tr>
<td><code>grab</code></td>
<td><code>excludelistgrabnew</code></td>
</tr>
</tbody>
</table>

- `update`:  
  - `imageupdate`: sync, image to running node  
- `install`:  
  - `node-provisioning`: full, image to starting node  
  - `node-provisioning`: node provisioning  
  - with installmode FULL  
- `grab`:  
  - `grabimage`: full, running node to image  
  - `cmsh`, `grabimage -i`/Grab to image
  - `grabimage`/Synchronize 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 `excludelistupdate` 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.
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 \texttt{fsmounts} mode can be excluded from being mounted on the nodes in the first place, by removing them from the listed mounts within the \texttt{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 \texttt{/etc/fstab} without using \texttt{cmsh} or Bright 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 \texttt{imageupdate}. Neglecting to do this may wipe out the filesystem, if it happens to be mounted in the middle of an \texttt{imageupdate} operation.

The \texttt{fstab} system is a statically mounting system, and not an auto-mounter: While \texttt{fstab} mounts filesystems automatically, system administrators should not confuse that with auto-mounting. Auto-mounting as provided by \texttt{autofs} is designed for the dynamic mounting of filesystems on demand by regular users. The \texttt{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.

\textbf{Editing An Exclude List}

A sample \texttt{cmsh} one-liner which opens up a text editor in a category so that the exclude list for updates can be edited is:

\begin{verbatim}
cmsh -c "category use default; set excludelistupdate; commit"
\end{verbatim}

Similarly, the exclude list for updates can also be edited in Bright View via the clickpath:

\textbf{Provisioning Modifications Via \texttt{excludelistmanipulatescript}}

Sometimes the administrator has a need to slightly modify the execution of exclude lists during provisioning. The \texttt{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 \texttt{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 \texttt{cmsh}:

\begin{verbatim}
[bright92]% device use node001
[bright92->device[node001]]% set excludelistmanipulatescript
(a vi session will start. A script is edited and saved)
[bright92->device[node001]]% commit
\end{verbatim}

The script can be as simple as:

\textbf{Example}

\begin{verbatim}
#!/bin/bash

echo "- *"
echo 'no-new-files: - *'
\end{verbatim}
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:

```bash
#!/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:

```
[bright92] device use node001
[bright92->device[node001]]% get excludelistmanipulatescript (the script is put in)
[bright92->device[node001*]]% commit; imageupdate
Performing dry run (use synclog command to review result, then pass -w to perform real update)...
[bright92->device[node001]]%
[bright92-] device[node001*] # This and next line added by category excludelistmanipulatescript.
```

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 /
```
Updating Running Nodes

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 wildcard 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 18.4.8.

- **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 `excludelistmanipulatescript` tool is powerful enough to do it, but it has some issues due to its power. For example, it is a script, which means that it is called whenever it is used, and so uses up some extra resources. Also, it is a bit tricky to set up.

  An easier way to manipulate exclude lists for the unneeded files and paths is via the `excludelistsnippets` tool, described in section 4.4.1 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

```
[bright92->category[default]]% set excludelistfullinstall
... a text editor such as vi opens up and the list can be edited ...
[bright92->category*[default*]]% commit
```

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

Example

```
[bright92->device[node001]]% set excludelistfullinstall
... a text editor such as vi opens up and the list can be edited ...
[bright92->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**

  `[bright92->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 244), 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**

    `[bright92->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:

    `[bright92->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 exclude_list 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 exclude_list get options correspond to the intentions of the associated exclude list types as distinguished on page 244.

5.6.2 Updating Running Nodes: With cmsh Using imageupdate

Using a defined exclude_list_update property (section 5.6.1), the imageupdate command of cmsh is used to start an update on a running node:

Example

[bright92->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 bright92: Provisioning started on node node001
[bright92->device]# imageupdate -n node001: image update in progress ...
[bright92->device]#
Tue Jan 11 12:13:44 2011 bright92: 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

[bright92->device]# imageupdate -n node001 -w
Provisioning started on node node001
node001: image update in progress ...
[bright92->device]# Provisioning completed on node node001

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

In Bright 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 Bright View automatically updates the provisioners first via the update_provisioners 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 provisioning_node_autoupdate_timeout 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 imagemupdate command therefore usually makes sense.

- By default, the cluster manager does not allow provisioning if automount (page 867) 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 imagemupdate 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 Bright View Add Functions

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

Example

[bright92->device]% add physicalnode node002 10.141.0.2
[bright92->device*[node002*]% commit

The Bright View equivalent of this is following the clickpath:
Devices→Nodes→Add→Settings→Hostname
then adding the value node002 to Hostname, and saving it.

When adding the node objects in cmsh and Bright View, some values (IP 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 is set to a static IP address that is set via cmsh or Bright View by the administrator.

Adding new node objects as “placeholders” can also be done from cmsh or Bright 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 Bright View as in the preceding text, there is also a Bright 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.

The node creation wizard sets IP addresses for the nodes. By default it starts IP addressing for the new nodes by guessing an appropriate value from the node names. The administrator can override this starting value in the main screen of the wizard by modifying it in the header section of the IP addresses column (figure 5.18).

![Node Creation Wizard: Setting Interfaces](image)

The cmsh equivalent of the node creation wizard is the foreach loop with the clone option acting on a node (section 2.5.5).

## 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 12.
  - On the head node, the most relevant software can be updated with `yum`, `zypper`, or `apt`, as explained in section 12.2. 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 12.4. 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@bright92 ~]# 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 Bright 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, table 3.2.1). New nodes are granted leases only after `lockdowndhcpd` is set to `no` in cmsh, or **Lock down dhcpd** is disabled in Bright 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.

  In such a case, removing all the connections and switches and just connecting the head node directly to a problem node, NIC-to-NIC, 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.

- **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 be 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.
5.8 Troubleshooting The Node Boot Process

- There may be 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.

In NVIDIA Bright Cluster Manager 9.0 onwards, 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\nheb-sun16 rhgb quiet net.ifnames=0 biosdevname=0"
```

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

```
[root@bright92 ~]# cp --preserve /boot/grub2/grub.cfg /boot/grub2/grub.cfg.orig
```
The GRUB configuration should be generated with:

```
[root@bright92 ~]# 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**

```
[bright92->softwareimage]% list
Name (key) Path
------------------ ------------------------------
default-image /cm/images/default-image
openstack-image /cm/images/openstack-image
```

```
[bright92->softwareimage]% use default-image
[bright92->softwareimage[default-image]% append kernelparameters " net.ifnames=0 biosdevname=0"
[bright92->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@bright92 ~]# 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@bright92 ~]# 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.
5.8 Troubleshooting The Node Boot Process

- 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

```
[bright92->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@bright92 ]# cmsh -c "softwareimage islocked"
Name             Locked
-------------- --------
default-image   yes
[root@bright92 ]# cmsh -c "softwareimage unlock default-image"
[root@bright92 ]# 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 12.7) 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 567.

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 17.7), to allow the viewing of console messages directly. A further depth in logging can be achieved by setting the kernel option \( \text{loglevel} = N \), 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 17.7) 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 17.7.4, 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 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-nsync`

### 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 12.4), 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 206). 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.
5.8 Troubleshooting The Node Boot Process

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 13:55:32 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). Partitions and/or filesystems are missing/corrupt. (Exit code 4, signal 0)
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
5.8 Troubleshooting The Node Boot Process

- 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

```
[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

```
fnd /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. mpt2sas in this case:

Example

```
[bright92]# softwareimage use default-image
[bright92->softwareimage[default-image]]% kernelmodules
[bright92->softwareimage[default-image]->kernelmodules]% add mpt2sas
[bright92->softwareimage[default-image]->kernelmodules[mpt2sas]]% commit
Thu May 19 16:54:52 2011 [notice] bright92: Initial ramdisk for image de\fault-image is being generated
[bright92->softwareimage[default-image]->kernelmodules[mpt2sas]]%
Thu May 19 16:55:43 2011 [notice] bright92: Initial ramdisk for image de\fault-image was regenerated successfully.
```

```
[bright92->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.

![No BMC Interface](image)

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-
The `failOnMissingBmc` 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@bright92 ~]# module load ipmitool
[root@bright92 ~]# 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@bright92 ~]# module load ipmitool
[root@bright92 ~]# 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@bright92 ~]# 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@bright92 ~]# 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@bright92 ~]# 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 bright92 systemd[1]: Starting IPMI Driver...
Dec 19 14:34:27 bright92 openipmi-helper[8930]: Startup failed.
Dec 19 14:34:27 bright92 systemd[1]: ipmi.service: main process exited, code=exited, status=1/\FAILURE
Dec 19 14:34:27 bright92 systemd[1]: Failed to start IPMI Driver.
Dec 19 14:34:27 bright92 systemd[1]: Unit ipmi.service entered failed state.
Dec 19 14:34:27 bright92 systemd[1]: ipmi.service failed.

Further details can be found in the journal:

Example

[root@bright92 ~]# journalctl -xe | grep -i ipmi
...
-- Unit ipmi.service has begun starting up.
Dec 19 14:34:27 bright92 kernel: ipmi message handler version 39.2
Dec 19 14:34:27 bright92 kernel: IPMI System Interface driver.
Dec 19 14:34:27 bright92 kernel: ipmi_si: Unable to find any System Interface(s)
Dec 19 14:34:27 bright92 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
5.8 Troubleshooting The Node Boot Process

[root@bright92 ~]# 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 bright92 systemd[1]: Starting IPMI Driver...
Dec 19 14:37:10 bright92 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@bright92 ~]# module load ipmitool
[root@bright92 ~]# ipmitool lan print
Set in Progress : Set Complete
Auth Type Support : MD5 PASSWORD
Auth Type Enable : Callback : MD5 PASSWORD
                  : User : MD5 PASSWORD
                  : Operator : MD5 PASSWORD
                  : Admin : MD5 PASSWORD
                  : OEM :
IP Address Source : Static Address
IP Address : 93.184.216.34
Subnet Mask : 255.255.255.0
MAC Address : aa:bb:01:02:cd:ef
SNMP Community String : public
IP Header : TTL=0x00 Flags=0x00 Precedence=0x00 TOS=0x00
BMC ARP Control : ARP Responses Enabled, Gratuitous ARP Disabled
Gratituous ARP Intrvl : 0.0 seconds
Default Gateway IP : 93.184.216.1
Default Gateway MAC : 00:00:00:00:00:00
Backup Gateway IP : 0.0.0.0
Backup Gateway MAC : 00:00:00:00:00:00
802.1q VLAN ID : Disabled
802.1q VLAN Priority : 0
RMCP+ Cipher Suites : 0,1,2,3,4,6,7,8,9,11,12,13,15,16,17,18
Cipher Suite Priv Max : caaaaaaaaaaaaaa
                  : X=Cipher Suite Unused
                  : c=CALLBACK
                  : u=USER
                  : o=OPERATOR
                  : a=ADMIN
                  : O=OEM

During normal operation the metrics (Appendix G) displayed by 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@bright92 ~]# module load ipmitool
[root@bright92 ~]# ipmitool sensor list all
<table>
<thead>
<tr>
<th>Sensor</th>
<th>Value</th>
<th>Unit</th>
<th>Status</th>
<th>Min</th>
<th>Max</th>
<th>Threshold</th>
</tr>
</thead>
<tbody>
<tr>
<td>Ambient Temp</td>
<td>22.000</td>
<td>°C</td>
<td>ok</td>
<td>38.000</td>
<td>41.000</td>
<td>45.000</td>
</tr>
<tr>
<td>AVG Power</td>
<td>300.000</td>
<td>Watts</td>
<td>ok</td>
<td>na</td>
<td>na</td>
<td>na</td>
</tr>
<tr>
<td>Fan 1 Tach</td>
<td>4125.000</td>
<td>RPM</td>
<td>ok</td>
<td>750.000</td>
<td>na</td>
<td>na</td>
</tr>
</tbody>
</table>
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 Bright View or cmsh, using profiles with sets of tokens.

6.1 Managing Users And Groups With Bright View

Within Bright 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 regular 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 an edit dialog (figure 6.2). These additions can be committed via the SAVE button.
When saving an addition or modification:

- User and group ID numbers are automatically assigned from UID and GID 1000 onwards.
- A home directory is created and a login shell is set. Users with unset passwords cannot log in.
Group management in Bright 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 Bright 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 Bright View of section 6.1. These functions are run from within the user mode of cmsh:

Example

```
[root@bright92 ~]# cmsh
[bright92]% user
[bright92->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

```
[bright92->user]% add maureen
[bright92->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

```
[bright92->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 Bright View standard display (figure 13.5).

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

```
[bright92->user*[maureen*]]% show
Parameter Value
-----------------------  -----------------------
Accounts
Managees
Name maureen
Primary group
Revision
Secondary groups
```
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

[bright92->user[maureen]]% commit
[bright92->user[maureen]]% show

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>Accounts</td>
<td></td>
</tr>
<tr>
<td>Managees</td>
<td></td>
</tr>
<tr>
<td>Name</td>
<td>maureen</td>
</tr>
<tr>
<td>Primary group</td>
<td>maureen</td>
</tr>
<tr>
<td>Revision</td>
<td></td>
</tr>
<tr>
<td>Secondary groups</td>
<td></td>
</tr>
<tr>
<td>ID</td>
<td>1001</td>
</tr>
<tr>
<td>Common name</td>
<td>maureen</td>
</tr>
<tr>
<td>Surname</td>
<td>maureen</td>
</tr>
<tr>
<td>Group ID</td>
<td>1001</td>
</tr>
<tr>
<td>Login shell</td>
<td>/bin/bash</td>
</tr>
<tr>
<td>Home directory</td>
<td>/home/maureen</td>
</tr>
<tr>
<td>Password</td>
<td>*********</td>
</tr>
<tr>
<td>email</td>
<td></td>
</tr>
<tr>
<td>Profile</td>
<td></td>
</tr>
<tr>
<td>Create cmjob certificate</td>
<td>no</td>
</tr>
<tr>
<td>Write ssh proxy config</td>
<td>no</td>
</tr>
<tr>
<td>Shadow min</td>
<td>0</td>
</tr>
<tr>
<td>Shadow max</td>
<td>999999</td>
</tr>
<tr>
<td>Shadow warning</td>
<td>7</td>
</tr>
<tr>
<td>Inactive</td>
<td>0</td>
</tr>
<tr>
<td>Last change</td>
<td>1970/1/1</td>
</tr>
<tr>
<td>Expiration date</td>
<td>2038/1/1</td>
</tr>
</tbody>
</table>

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 on 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**

```
[bright92->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
Continuing the session from the end of section 6.2.2, the password can be set at the user context prompt like this:

Example

```
[bright92->user[maureen]]% set password seteca5tr0n0my
[bright92->user*[maureen*]]% commit
[bright92->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

```
[bright92->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@bright92 ~]# #first create the LDAP salted SHA-1 hash of the password:
[root@bright92 ~]# /cm/local/apps/openldap/sbin/slappasswd -h {SSHA} -s seteca5tr0n0my
[root@bright92 ~]# {SSHA}sViD+lfSTtllyO9u02Q56d
[root@bright92 ~]# cmsh
[bright92%] user use maureen
[bright92->user[maureen]]% set password enter new password: #here and in the next line {SSHA}sViD+lfSTtllyO9u02Q56d is typed in
retype new password: [bright92->user[maureen]]% commit
[bright92->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 Warning: Permanently added 'node001' (ECDSA) to the list of known hosts.
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 freds are now added to the LDAP directory, along with a group `printer`:

```
[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.
Example

[bright92->user[maureen]]% add tim; add fred
[bright92->user*[fred*]]% exit; group; add printer
[bright92->group*[printer*]]% commit
[bright92->group[printer]]% exit; exit; user
[bright92->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 onwards, 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

[bright92->user*]% commit
Successfully committed 2 Users
[bright92->user*] % group use printer
[bright92->group[printer]]% append members maureen tim fred; commit
[bright92->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:

[bright92->group[printer]]% removefrom members tim fred; commit
[bright92->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

[bright92->group[printer]]% clear members
[bright92->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

```
[bright92->group*][printer*]]% refresh
[bright92->group[printer]]% show
Parameter  Value
----------  ------------
ID         1002
Revision   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

```
[bright92->group[printer]]% user use maureen
[bright92->user[maureen]]% remove -d; commit
Successfully removed 1 Users
Successfully committed 0 Users
[bright92->user]% !ls -d /home/* | grep maureen  #no maureen left behind
```

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, the `/etc/nslcd.conf`, `/etc/openldap/ldap.conf`, and the certificate files under `/cm/local/apps/openldap/etc/certs/` should be copied over. 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 13.

- 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@bright92 ~]# cmsh
  [bright92]# cmsh
  [bright92->user]# add ldapcheck; commit
  [bright92->user[ldapcheck]]# monitoring setup use ldap
  [bright92->monitoring->setup[ldap]]# show
  Parameter | Value
  ----------------- --------------------------
  ...        
  Arguments
  ...
  [bright92->monitoring->setup[ldap]]# set arguments "ldapcheck"; commit
  [bright92->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 Bright View or `cmsh`.

- Bright 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@bright92 ~]# 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`.  

---

### Chapter 13

Health checks and health check objects are discussed in Chapter 13.
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`. 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 Bright 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 Bright 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 639 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:

```bash
index entryCSN eq
index entryUUUID 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 275 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 CM-Daemon 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 275 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 275 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 syncrep1 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
Tokens can be assigned by the administrator to users so that users can carry out some of the operations that the administrator do with Bright 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:

<table>
<thead>
<tr>
<th>Profile name</th>
<th>Default Tasks Allowed</th>
</tr>
</thead>
<tbody>
<tr>
<td>admin</td>
<td>all tasks</td>
</tr>
<tr>
<td>cloudjob</td>
<td>cloud job submission</td>
</tr>
<tr>
<td>cmhealth</td>
<td>health-related prejob tasks</td>
</tr>
<tr>
<td>cmpam</td>
<td>the cluster manager PAM tasks</td>
</tr>
<tr>
<td>litenode</td>
<td>CMDaemon Lite (section 2.6.7) tasks</td>
</tr>
<tr>
<td>monitoringpush</td>
<td>pushing raw monitoring data to CMDaemon via a JSON POST (page 404 of the Developer Manual)</td>
</tr>
<tr>
<td>node</td>
<td>node-related</td>
</tr>
<tr>
<td>portal</td>
<td>user portal viewing</td>
</tr>
<tr>
<td>power</td>
<td>device power</td>
</tr>
<tr>
<td>readonly</td>
<td>view-only</td>
</tr>
</tbody>
</table>

The available preconfigured profiles in cmsh can be seen by running the list command in profile mode. Another way to see them is by using tab-completion prompting, as follows:

Example

[root@bright92 ~]# cmsh
[bright92]% profile
[bright92->profile]% show <TAB> <TAB>
admin cloudjob cmhealth cmpam litenode monitoringpush node portal power readonly

The tokens, and other properties of a particular profile can be seen within profile mode as follows:

Example

[bright92->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

[bright92->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@bright92 ~]# cmsh
[bright92]% user use conner
[bright92->user[conner]]% get profile
[bright92->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@bright92 ~]# cmsh
[bright92]% profile use readonly
[...[readonly]]% removefrom tokens burn_status_token get_burn_log_token
[bright92]%->profile*[readonly*]% commit
```

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

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

![Profile Management](image)

Figure 6.3: Bright View Profile Token Management

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 Bright 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 283) or Bright View (page 284) is described later on. After creating such a certificate, the openssl utility can be used to examine its structure and properties. In the following example most of the output has been elided in order to highlight the expiry
date (30 days from the time of generation), the common name (democert), the key size (2048), profile properties (readonly), and system login name (peter), for such a certificate:

[root@bright92]# openssl x509 -in peterfile.pem -text -noout

Certificate:
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@bright92]#

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@bright92 ~]# cmsh -c "cert; listcertificates"

<table>
<thead>
<tr>
<th>Serial num</th>
<th>Days left</th>
<th>Profile</th>
<th>Country</th>
<th>Name</th>
<th>Revoked</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>36451</td>
<td>admin</td>
<td>US</td>
<td>Administrator</td>
<td>Yes</td>
</tr>
<tr>
<td>10</td>
<td>9</td>
<td>readonly</td>
<td>ef</td>
<td>democert</td>
<td>Yes</td>
</tr>
<tr>
<td>11</td>
<td>36496</td>
<td>node</td>
<td>NL</td>
<td>52-54-00-de-e3-6b</td>
<td>No</td>
</tr>
<tr>
<td>12</td>
<td>36496</td>
<td>node</td>
<td>NL</td>
<td>52-54-00-44-fb-85</td>
<td>No</td>
</tr>
<tr>
<td>13</td>
<td>36496</td>
<td>admin</td>
<td>UK</td>
<td>otheradmin</td>
<td>No</td>
</tr>
</tbody>
</table>

Listing certificates with Bright View: The Bright View equivalent for listing certificates is via the click-path Identity Management → Certificates (figure 6.4):

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.

Custom certificates are also listed in the certificate lists.

Creating A Custom Certificate

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

- Bright View: via the clickpath Identity Management → Users → Edit → Profile to set the Profile.
Creating a new certificate for \texttt{cmsh} users: Creating a new certificate in \texttt{cmsh} is done from \texttt{cert} mode using the \texttt{createcertificate} command, which has the following help text:

\texttt{[bright92->cert] \% help createcertificate}

Name:
\texttt{createcertificate - Create a new certificate}

Usage:
\texttt{createcertificate <key-length> <common-name> <organization> <organizational-unit> <locality> <state> <country> <profile> <sys-login> <days> <key-file> <cert-file>}

Arguments:
\begin{itemize}
\item \texttt{key-file}
  
  Path to key file that will be generated
\item \texttt{cert-file}
  
  Path to pem file that will be generated
\end{itemize}

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 \texttt{peter}, can be created with:

\textbf{Example}

\begin{verbatim}
createcertificate 2048 democert a b c d ef readonly peter 30 /home/peter/peterfile.key /home/peter/peterfile.pem
\end{verbatim}

Fri Aug 22 06:18:27 2014 [notice] bright92: New certificate request with ID: 1
[\texttt{bright92->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 certificates are owned by the owner generating them, so they are root-owned if \texttt{root} was running \texttt{cmsh}. This means that user \texttt{peter} cannot use them until their ownership is changed to user \texttt{peter}:

\textbf{Example}

\begin{verbatim}
[root@bright92 ~] \# cd /home/peter
[root@bright92 peter] \# ls -l peterfile.*
\end{verbatim}
Other users must have the certificate ownership changed to their own user names. 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@bright92 ~] cmsh -i peterfile.pem -k peterfile.key
   [bright92]% 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 Bright View users: As in the case of cmsh, a Bright 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 Bright View use this certificate for their Bright 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:
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> [...]
```

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 read-only 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 284.

### 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.
Workload Management

For clusters that have many users and a significant load, a workload manager (WLM) system allows a more efficient use of resources to be enforced for all users than if there were no such system in place. This is because without resource management, there is a tendency for each individual user to over-exploit common resources.

When a WLM is used, the end user can submit a job to it. This can be done interactively, but it is typically done as a non-interactive batch job.

The WLM assigns resources to the job, and checks the current availability as well as checking its estimates of the future availability of the cluster resources that the job is asking for. The WLM then schedules and executes the job based on the assignment criteria that the administrator has set for the WLM system. After the job has finished executing, the job output is delivered back to the user.

Among the hardware resources that can be used for a job are GPUs. Installing CUDA software to enable the use of GPUs is described in section 7.4 of the *Installation Manual*. Configuring GPU settings for the cluster manager is described in section 3.13.2 of the *Administration Manual*. Configuring GPU settings for an individual WLM is described in the section on getting that particular WLM up and running.

The details of job submission from a user’s perspective are covered in the *User Manual*.

Sections 7.1–7.5 cover the installation procedure to get a WLM up and running.

Sections 7.6–7.7 describe how Bright View and *cmsh* are used to view and handle jobs, queues and node drainage.

Section 7.8 shows examples of WLM assignments handled by the cluster manager.

Section 7.9 describes the power saving features of WLMs.

Section 7.10 describes *cgroups*, a resources limiter, mostly in the context of WLMs.

Section 7.11 describes WLM customizations for settings other than the common settings covered by the cluster manager.

### 7.1 Workload Managers Choices

Some WLM packages are installed by default, others require registration from the distributor before installation.

During cluster installation, a WLM can be chosen (figure 3.10 of the *Installation Manual*) for setting up. The choices are:

- **PBS**: An HPC job scheduler, originally developed at NASA, now developed by Altair. This is integrated with the cluster manager in these variants:

  1. **PBS Pro version 2021**: A commercial variant, with commercial support from Altair.
  2. **OpenPBS version 20**: This is a community-supported variant. The variant was known as PBS Pro CE before version 20.

- **Slurm v19 and v20**: A free (GPL) job scheduler, with commercial support.
**Univa Grid Engine (UGE), v8.6.0:** This is a further development of the older Sun Grid Engine. UGE is proprietary commercial software, developed by Univa. NVIDIA Bright Cluster Manager 9.2 is intended to support integration with UGE versions 8.6.0 and higher, assuming no major design changes in UGE.

**LSF v10.1:** IBM Spectrum LSF (Load Sharing Facility) version 10.1, is a further development of what used to be IBM Platform LSF. It is now a part of the IBM Spectrum LSF Suite v10.2.0.9. The cluster manager no longer supports deployment of IBM Spectrum LSF suite. LSF Standard Edition remains the supported option.

**None:** For clusters that need no HPC job-scheduling.

The WLMs in the preceding list can also be chosen and set up later using the `cm-wlm-setup` tool (section 7.3).

After installation, if there are no major changes in the WLM for updated versions of the workload managers, then

- WLMs that are packaged with the cluster manager (Slurm, PBS) can have their packages updated using standard package update commands (`yum update` and similar). The installation and configuration of the WLM from the updated packages is carried out as described later on in this chapter.
- WLMs that are installed by picking up software from the vendor (UGE, LSF) can be updated by following vendor guidelines. For UGE the installation is carried out as outlined in section 7.5.2.

### 7.2 Forcing Jobs To Run In A Workload Management System

Another preliminary step is to consider forcing users to run jobs only within the WLM system. Having jobs run via a WLM is normally a best practice.

For convenience, the cluster manager defaults to allowing users to log in via `ssh` to a node, using the authorized keys files stored in each users directory in `/home` (section 2.3.2). This allows users to run their processes without restriction, that is, outside the WLM system. For clusters with a significant load this policy results in a sub-optimal use of resources, since such unplanned-for jobs disturb any already-running jobs.

Disallowing user logins to nodes, so that users have to run their jobs through the WLM system, means that jobs are then distributed to the nodes only according to the planning of the WLM. If planning is based on sensible assignment criteria, then resources use is optimized—which is the entire aim of a WLM in the first place.

#### 7.2.1 Disallowing User Logins To Regular Nodes Via `cmsh`

The `usernodelogin` setting of `cmsh` restricts direct user logins from outside the WLM, and is thus one way of preventing the user from using node resources in an unaccountable manner. The `usernodelogin` setting is applicable to node categories only, rather than to individual nodes.

In `cmsh` the attribute of `usernodelogin` is set from within `category mode`:

**Example**

```
[root@bright92 ~]# cmsh
[bright92%] category use default
[bright92->category[default]]% set usernodelogin onlywhenjob
[bright92->category[default]]% commit
```

The attributes for `usernodelogin` are:

- **always** (the default): This allows all users to `ssh` directly into a node at any time.
- **never**: This allows no user other than root to directly `ssh` into the node.
7.2 Forcing Jobs To Run In A Workload Management System

- **onlywhenjob**: This allows the user to `ssh` directly into the node when a job is running on it. It typically also prevents other users from doing a direct `ssh` into the same node during the job run, since typically the WLM is set up so that only one job runs per node. However, an `ssh` session that is already running is not automatically terminated after the job is done.

  Some cluster administrators may wish to allow some special user accounts to override the `onlywhenjob` setting, for example for diagnostic purposes. Before giving the details of how to override the setting, some background explanation is probably useful:

  The `onlywhenjob` setting works with the PAM system, and adds the following line to `/etc/pam.d/sshd` on the regular nodes:

  ```
  account required pam_bright.so
  ```

  Nodes with the `onlywhenjob` restriction can be configured to allow a particular set of users to access them, despite the restriction, by whitelisting them in the PAM system, as follows:

  Within the software image `<node image>` used by the node, that is under `/cm/images/<node image>`, the administrator can add the set of user accounts to the file `/etc/security/pam_bright.d/pam_whitelist.conf`. This file is installed in the software image with a chroot installation (section 12.4) of the `cm-libpam` package.

  Other adjustments to PAM configuration, such as the number of attempted logins and the associated wait time per login, can be carried by editing the `/etc/security/pam_bright.d/cm-check-alloc.conf` file.

  The image can then be updated to implement the whitelist, by running the `imageupdate` command in `cmsh` (section 5.6.2), or by clicking the `Update node` option in Bright View (section 5.6.3).

7.2.2 Disallowing User Logins To Regular Nodes Via Bright View

In Bright View, user node login access is set via a category setting, for example for the `default` category via the clickpath in figure 7.1:

Grouping → Categories[default] → Edit → Settings → User node login

![Figure 7.1: Disallowing User Logins To Nodes Via Bright View](image-url)
7.2.3 Disallowing Other User Processes Outside Of Workload Manager User Processes

Besides disabling user logins, administrators may choose to disable interactive jobs in the WLM as an additional measure to prevent users from starting jobs on other nodes.

Administrators may also choose to set up scripts that run after job execution. Such scripts can terminate user processes outside the WLM, as part of a policy, or for general administrative hygiene. These are Epilog scripts and are part of the WLM.

The WLM documentation has more on configuring these options.

7.2.4 High Availability By Workload Managers

NVIDIA Bright Cluster Manager uses the existing built-in high availability (HA) functionalities of workload managers as much as possible. A double server HA configuration with the existing built-in HA functionality can be carried out using the \texttt{cm-wlm-setup} utility (section 7.3), or using the Bright View HA wizard (section 7.4.1). The built-in functionality makes use of the primary WLM server and secondary WLM server, which are placed on separate nodes.

The HA configuration aspect (section 18.1.3) in this case means that a WLM server role is assigned to both the WLM primary and the WLM secondary nodes. These primary WLM and secondary WLM servers can then:

- both be on head nodes,
- both be on regular (compute) nodes

It is not possible to configure a mixed setup—that is with one head node and one regular node—for the WLM servers on an HA setup.

If the head nodes are configured with the WLM server roles, then in the cluster entity configuration the \texttt{primary server} value is empty. This is because NVIDIA Bright Cluster Manager always configures the primary WLM server on the primary cluster head node.

If two regular nodes are configured with the WLM server roles instead, then in the cluster entity configuration the \texttt{primary server} parameter is set to one of the compute nodes.

The cluster primary head node and the WLM primary server should not be confused. In particular, the active cluster head node and active workload manager server are not necessarily the same for the case of the Slurm, PBS Pro, or LSF workload managers. For these, if the passive cluster head node becomes an active cluster head node without a crash, then this does not trigger a passive WLM server on the newly active node to also become an active WLM server. Thus, the WLM server can be active on a passive head node, and vice versa, because the WLM primary server is independent of the cluster primary head node.

7.3 Installation Of Workload Managers

Normally the administrator selects a WLM to be used during the cluster manager installation (figure 3.10 of the \textit{Installation Manual}). A WLM may however also be added and configured after the cluster manager has been installed, using \texttt{cm-wlm-setup}, or the Bright View WLM wizard.

With most other objects, the cluster manager front ends—\texttt{cmsh} and Bright View—can be used to create a new object from scratch, or can clone a new object from another existing object. However, the front ends cannot do this for a WLM object. An attempt to create or clone a new WLM object via \texttt{cmsh} or Bright View is prohibited by the front ends, because there are many pitfalls possible in configuration.

A new WLM object, and WLM instance, can therefore only be installed via \texttt{cm-wlm-setup}, the Bright View WLM wizard, or by selecting a WLM during the initial cluster manager installation.

7.3.1 Running \texttt{cm-wlm-setup} In CLI Mode

The recommended way to run the \texttt{cm-wlm-setup} utility is without options or arguments, in which case an Ncurses dialog starts up. An Ncurses run with \texttt{cm-wlm-setup} is covered in section 7.3.2.
However, the `cm-wlm-setup` utility can alternatively be used in a non-Ncurses, command-line, mode, with options and arguments. The utility has the following usage:

```
[root@bright92 ~]# cm-wlm-setup -h
usage: Workload manager setup cm-wlm-setup [-c <config_file>]
        [--setup | --disable]
        [--wlm <name>]
        [--server-nodes SERVER_NODES]
        [--server-primary SERVER_PRIMARY]
        [--server-overlay-name SERVER_OVERLAY_NAME]
        [--server-overlay-priority SERVER_OVERLAY_PRIORITY]
        [--client-categories CLIENT_CATEGORIES]
        [--client-nodes CLIENT_NODES]
        [--client-overlay-name CLIENT_OVERLAY_NAME]
        [--client-overlay-priority CLIENT_OVERLAY_PRIORITY]
        [--client-slots <slots>]
        [--submit-categories SUBMIT_CATEGORIES]
        [--submit-nodes SUBMIT_NODES]
        [--submit-overlay-name SUBMIT_OVERLAY_NAME]
        [--submit-overlay-priority SUBMIT_OVERLAY_PRIORITY]
        [--wlm-cluster-name WLM_CLUSTER_NAME]
        [--reboot] [--reset-cgroups]
        [yes-i-really-mean-it]
        [--archives-location <path>]
        [--license <license>] [--purge]
        [--accounting-overlay-name ACCOUNTING_OVERLAY_NAME]
        [--accounting-overlay-priority ACCOUNTING_OVERLAY_PRIORITY]
        [-v] [-no-distro-checks] [-json]
        [--output-remote-execution-runner]
        [--on-error-action debug,remotedebug,undo,abort]
        [--skip-packages]
        [--min-reboot-timeout <reboot_timeout_seconds>]
        [--dev] [-h]
```

The help output from running `cm-wlm-setup -h` continues on beyond the preceding text output, and presents more options.

These options can be grouped as follows:

**Optional Arguments**
- `--setup`: Helps set up a server, enable roles, and create the default queues/partitions
- `--disable`: Disable WLM services
- `-h`, `--help`: Displays the help screen

**Common Arguments**
- `-c <YAML configuration file>`: Loads a runtime configuration for plugins, from a YAML configuration file.

**Options For Installing Or Managing A WLM**
- `--wlm <WLM name>`: Specifies which WLM is to be set up. Choices for `<WLM name>` are:
  - openpbs
  - pbspro
  - slurm
- uge
- lsf

- --wlm-cluster-name <WLM cluster name>: Specifies the name for the new WLM cluster that is to be set up.
- --reboot: Reboot after install

Server Role Settings
- --server-nodes <server nodes>: Sets the server roles of the WLM to the value set for <server nodes>, which is a comma-separated list of nodes. Default value: HEAD, which is a reserved name for the head node.
- --server-primary <primary server>: Sets the hostname used for the primary server to <primary server>. Default name: HEAD.
- --server-overlay-name <server overlay name>: Sets the server role configuration overlay name to <server overlay name>. Default name: <WLM name>-server, where <WLM name> is the name specified in the --wlm option.
- --server-overlay-priority <server overlay priority>: Sets the server role configuration overlay priority to <server overlay priority>. Default value: 500.

Client Role Settings
- --client-categories <client categories>: Sets the client roles of the WLM to the value set for <client categories nodes>, which is a comma-separated list of node categories. Default value: default.
- --client-nodes <client nodes>: Sets the client roles of the WLM to the value set for <client nodes>, which is a comma-separated list of nodes. No value set by default.
- --client-overlay-name <client overlay name>: Sets the client role configuration overlay name to <client overlay name>. Default name: <WLM name>-client, where <WLM name> is the name specified in the --wlm option.
- --client-overlay-priority <client overlay priority>: Sets the client role configuration overlay priority to <client overlay priority>. Default value: 500.
- --client-slots <slots>: Sets the number of slots on the client to <slots>.

Submit Role Settings
- --submit-categories <submit categories>: Sets the submit roles of the WLM to the value set for <submit categories nodes>, which is a comma-separated list of node categories that are submit nodes. Default value: default.
- --submit-nodes <submit nodes>: Sets the submit roles of the WLM to the value set for <submit nodes>, which is a comma-separated list of submit nodes. No value set by default.
- --submit-overlay-name <submit overlay name>: Sets the submit role configuration overlay name to <submit overlay name>. Default name: <WLM name>-submit, where <WLM name> is the name specified in the --wlm option.
- --submit-overlay-priority <submit overlay priority>: Sets the submit role configuration overlay priority to <submit overlay priority>. Default value: 500.

Disable Options
- --reset-cgroups: Disable joining cgroup controllers with systemd setting JoinControllers
- --yes-i-really-mean-it: Required for additional safety
7.3 Installation Of Workload Managers

Workload Manager Specific Options

- **--archives-location** *<path>*: Set the directory path for the archive files, only for LSF and UGE. This parameter is mandatory for LSF installation.
- **--purge**: Remove the directories on disable, for UGE and LSF.
- **--license** *<path>*: Set the path to the PBSPro or LSF license.
- **--accounting-overlay-name** *<accounting overlay name>*: Sets the accounting role configuration overlay.

Advanced Options

- **-v, --verbose**: This displays a more verbose output. It can be helpful in troubleshooting.
- **--no-distro-checks**: Disables distribution checks based on ds.json.
- **--json**: Use json formatting for logs printed to STDOUT.
- **--output-remote-execution-runner**: Format output for CMDaemon.
- **--on-error-action** {debug, remotedebug, undo, abort}: Upon encountering a critical error, instead of asking the user for choice, the setup will do the selected action.
- **--skip-packages**: Skip the stages which install packages. Requires packages to be already installed.
- **--min-reboot-timeout** *<timeout>*: How long to wait for nodes to finish reboot, in seconds. Minimum value: 300. Default value: 300.
- **--dev**: Enables additional command line arguments for developers.

For example, if UGE packages are downloaded to /root/ugedl, then a UGE setup could be carried out with (much output elipsized):

**Example**

```
[bright92 ~]# cm-wlm-setup -v --setup --wlm uge --archives-location /root/ugedl --client-categories default
I 19-12-05 15:29:46 | Please wait...
D 19-12-05 15:29:46 | ##########################################################################
D 19-12-05 15:29:46 | ######################## START ###########################################
D 19-12-05 15:29:46 | ##########################################################################
D 19-12-05 15:29:46 | ### Logging Current Runtime environment
...
I 19-12-05 15:33:20 | Took: 00:33 min.
I 19-12-05 15:30:20 | Progress: 100/100
I 19-12-05 15:30:20 | Finished execution for 'Workload manager setup'. status: completed
I 19-12-05 15:30:20 |
I 19-12-05 15:30:20 | Workload manager setup finished!
I 19-12-05 15:30:20 |
D 19-12-05 15:30:20 |
D 19-12-05 15:30:20 | END ##########################################################################
D 19-12-05 15:30:20 |
I 19-12-05 15:30:20 |
```
A check to see if the UGE instance (WLM cluster) has been created could be:

Example

```
[bright92 ~]# cmsh -c "wlm; list"
```

<table>
<thead>
<tr>
<th>Type</th>
<th>Name (key)</th>
<th>Server nodes</th>
<th>Submit nodes</th>
<th>Client nodes</th>
</tr>
</thead>
<tbody>
<tr>
<td>-----</td>
<td>------------</td>
<td>--------------</td>
<td>--------------</td>
<td>--------------</td>
</tr>
<tr>
<td>UGE</td>
<td>uge</td>
<td>bright92</td>
<td>bright92,node001,node002 node001,node002</td>
<td></td>
</tr>
</tbody>
</table>

The UGE instance can then be removed with:

Example

```
[bright92 ~]#
cm-wlm-setup -v --disable --wlm uge --purge --yes-i-really-mean-it --wlm-cluster-name=uge
```

A log file of the setup can be seen at `/var/log/cm-wlm-setup.log`.

### 7.3.2 Running `cm-wlm-setup` In Ncurses Mode

Running `cm-wlm-setup` with no options and with no arguments brings up an Ncurses screen (figure 7.2).

![cm-wlm-setup Ncurses initial screen](image)

**Figure 7.2: cm-wlm-setup Ncurses initial screen**

**Express Installation**

The **Setup (Express)** menu option allows the administrator to select the workload manager in the next screen (figure 7.3), and to install it with a minimal number of configuration steps. If it has already been installed, but disabled via `cm-wlm-setup`, then it can also be re-enabled, instead of installed from scratch.

![cm-wlm-setup Ncurses WLM selection screen](image)

**Figure 7.3: cm-wlm-setup Ncurses WLM selection screen**
7.3 Installation Of Workload Managers

Step By Step Installation
If the Setup (Step By Step) menu option is chosen instead of the express option, then this also allows the administrator to select the workload manager in figure 7.3. But after selection, there are a number of extra configuration steps that can be carried out which are not available in the express configuration. Guidance is given for these extra steps, and sensible default values are already filled in for many options.

GPU Configuration Screens
Some of the extra steps cover GPU configuration screens. Those steps are discussed on page 7 of the Installation Manual.

Disabling An Installation
The Disable option in figure 7.2 allows the administrator to disable an existing instance.

Summary Screen
The screen that appears after the configuration steps are completed, is the Summary screen (figure 7.4). This screen allows the configuration to be viewed, saved, or saved and deployed.

Figure 7.4: cm-wlm-setup Ncurses summary screen

If deployment is carried out, then several screens of output are displayed. After the deployment is completed, the log file can be viewed at /var/log/cm-wlm-setup.log.

7.3.3 Configuration Of Slurm To Run Containerized Jobs Using Pyxis With Enroot

What Is Enroot?
As the README file for Enroot says, Enroot is an open source tool to turn container images into unprivileged sandboxes. Enroot can be thought of as an enhanced unprivileged chroot. It uses user and mount namespaces, as well as other modern kernel features, in order to create such sandboxes. It uses the same underlying technologies as containers, but removes much of the isolation that they inherently provide, while preserving filesystem separation. Further details on Enroot can be found at https://github.com/NVIDIA/enroot.

Enroot can be used with different workload managers, but for now only Slurm has been tightly integrated.

What Is Pyxis?
Pyxis (https://github.com/NVIDIA/pyxis) is a SPANK plugin for Slurm. SPANK (Slurm Plug-in architecture for Node and job Kontrol, man (8) spank) is a generic interface for job launch code control in Slurm. The plugin allows the user’s jobs to be executed seamlessly over Enroot in unprivileged containers. The plugin enables the Slurm submission utilities to provide container-related command line options.
**How Are Enroot And Pyxis Set Up In The Cluster Manager?**

Pyxis and Enroot can be set up by the administrator by choosing the appropriate options when Slurm is set up. In order to choose the appropriate options, cm-wlm-setup must be run in step by step mode (page 295).

This eventually presents a screen where the Pyxis setup can be configured:

![cm-wlm-setup Pyxis setup screen](image)

Figure 7.5: cm-wlm-setup Pyxis setup screen

The Pyxis screen is available for RHEL8-based systems and Ubuntu 20 and beyond. Older systems are not supported.

If yes is selected in the Pyxis setup screen then cm-wlm-setup installs the enroot and enroot+caps packages from https://github.com/NVIDIA/enroot/releases into the software images where the Slurm client role is to be assigned. It also installs them directly on the head node if the head node was selected to run jobs. Pyxis sources are downloaded from GitHub, compiled with the installed Slurm, and installed in the appropriate Slurm directory.

When Pyxis is set up, cm-wlm-setup also prepares the following configuration files for the compute nodes:

- `/etc/enroot/enroot.conf`: This is a symlink to `/cm/shared/apps/slurm/var/etc/enroot.conf`. The configuration file provides reasonable default settings that allows Enroot to be used by many users. Important settings in the file are:
  - `ENROOT_RUNTIME_PATH`: working directory for enroot, created per user. Default value: `/run/enroot/runtime/$(id -u)`
  - `ENROOT_CACHE_PATH`: directory where container layers are stored. Default value: `/run/enroot/cache/$(id -u)`
  - `ENROOT_DATA_PATH`: directory where the filesystems of running containers are stored. Default value: `/run/enroot/data/$(id -u)`
  - `ENROOT_SQUASH_OPTIONS`: options passed to `mksquashfs` to produce container images. Default value: `-noI -noD -noF -noX -no-duplicates`
  - `ENROOT_MOUNT_HOME`: mount the current user’s home directory by default. Default value: `yes`.

The administrator can change the symlink, or replace the file with a customized `enroot.conf`, if other values are preferred.


- `/cm/local/apps/slurm/var/prologs/50-prolog-enroot.sh`: symlink to `/cm/shared/apps/slurm/var/cm/prolog-enroot.sh`. This is the `slurmd` prolog that creates appropriate directories, with appropriate user permissions, that are used by Enroot.

- `/cm/local/apps/slurm/var/epilogs/50-epilog-enroot.sh`: symlink to `/cm/shared/apps/slurm/var/cm/epilog-enroot.sh`. Cleans the user directories used by Enroot.
When cm-wlm-setup finishes its Pyxis configuration run, then there is no need for the nodes to be rebooted. The plugin works immediately. The simplest way to validate the Pyxis/Enroot setup after Slurm setup is to try out an srun command:

Example

```
[user@bright92 ~]$ module load slurm
[user@bright92 ~]$ srun --container-image=ubuntu grep PRETTY /etc/os-release
pyxis: importing docker image: ubuntu
PRETTY_NAME="Ubuntu 20.04.3 LTS"
```

The new Slurm submission command option names start with either --container or --no-container. The full list of options can be displayed with the --help option. For example:

Example

```
[root@bright92 ~]$ srun --help | grep container
--container Path to OCI container bundle
--container-image=[USER@][REGISTRY#]IMAGE[:TAG][PATH]
    [pyxis] the image to use for the container
--container-mounts=SRC:DST[:FLAGS][,SRC:DST...]
    [pyxis] bind mount[s] inside the container. Mount
--container-workdir=PATH
    [pyxis] working directory inside the container
--container-name=NAME
    [pyxis] name to use for saving and loading the
        container on the host. Unnamed containers are
        containers are not. If a container with this name
        already exists, the existing container is used and
--container-save=PATH
        [pyxis] Save the container state to a squashfs
--container-mount-home
        [pyxis] bind mount the user's home directory.
--no-container-mount-home
--container-remap-root [pyxis] ask to be remapped to root inside the
        container. Does not grant elevated system
--no-container-remap-root
--container-entpoint [pyxis] execute the entpoint from the container
--no-container-entpoint
        container image
--container-writable [pyxis] make the container filesystem writable
--container-readonly [pyxis] make the container filesystem read-only
```

### 7.3.4 Prolog And Epilog Scripts

**What Prolog And Epilog Scripts Do**
The workload manager runs prolog scripts before job execution, and epilog scripts after job execution. The purpose of these scripts can include:

- Checking if a node is ready before submitting a job execution that may use it
- Preparing a node in some way to handle the job execution
- Cleaning up resources after job execution has ended.

The administrator can run custom prolog or epilog scripts for the queues from CMDaemon for UGE, or LSF, by setting such scripts in the Bright View or cmsh front ends.

Example
For PBS and Slurm, there are global prolog and epilog scripts, but editing them is not recommended. Indeed, in order to discourage editing them, the scripts cannot be set via the cluster manager front ends. Instead the scripts must be placed by the administrator in the software image, and the relevant nodes updated from the image.

**Detailed Workings Of Prolog And Epilog Scripts**

Even though it is not recommended, some administrators may nonetheless wish to link and edit the scripts directly for their own needs, outside of the Bright View or cmsh front ends. A more detailed explanation of how the prolog scripts work therefore follows:

When a workload manager is configured via `cm-wlm-setup` or via the Bright View setup wizard, then the workload manager is configured to run the generic prolog located in `/cm/local/apps/cmd/scripts/prolog`, and the generic epilog located in `/cm/local/apps/cmd/scripts/epilog`. The generic prolog and epilog scripts call a sequence of scripts for a particular workload manager in special directories. The directories have paths in the format:

1. `/cm/local/apps/<workload manager>/var/prologs/`
2. `/cm/local/apps/<workload manager>/var/epilogs/`

In these directories, scripts are stored with names that have suffixes and prefixes associated with them that make them run in special ways, as follows:

- **Suffixes used in the prolog/epilog directory:**
  - `-prejob` script runs prior to all jobs
  - `-cmjob`: script runs prior to job run in a cloud

- **Prefixes used in the prolog/epilog directory:**
  - `00-` to `99-`

  Number prefixes determine the order of script execution, with scripts with a lower number running earlier.

The script names can therefore look like:

**Example**

- `01-prolog-prejob`
- `10-prolog-cmjob`

Return values for the prolog/epilog scripts have these meanings:

- **0**: the next script in the directory is run.
- **A non-zero return value**: no further scripts are executed from the prolog/epilog directory.

Often, the script in a prolog/epilog directory is not a real script but a symlink, with the symlink going to a real file located in a different directory. The general script is then able to take care of what is expected of the symlink. The name of the symlink, and destination file, usually hints at what the script is expected to do.
7.3 Installation Of Workload Managers

For example, if any health checks are marked to run as prejob checks during cm-wlm-setup configuration, then each of the PBS workload manager variants use the symlink 01-prolog-prejob within the prolog directory `/cm/local/apps/<workload manager>/var/prologs/`. The symlink links to the script `/cm/local/apps/cmd/scripts/prolog-prejob`. In this case, the script is expected to run prior to the job.

Example

```
[root@bright92 apps]# pwd
/cm/local/apps
[root@bright92 apps]# ls -l *pbs*/var/prologs/
openpbs/var/prologs/:
  total 0
  lrwxrwxrwx 1 root root 40 Dec 23 15:44 01-prolog-prejob -> /cm/local/apps/cmd/scripts/prolog-prejob
pbspro/var/prologs/:
  total 0
  lrwxrwxrwx 1 root root 40 Dec 23 15:44 01-prolog-prejob -> /cm/local/apps/cmd/scripts/prolog-prejob
```

Epilog scripts (which run after a job run) have the location `/cm/local/apps/<workload manager>/var/epilogs/`. Epilog script names follow the same execution sequence pattern as prolog script names.

It should be noted that the 01-prolog-prejob symlink is created and removed by the cluster manager on each compute node where prejob is enabled in the workload manager entity. Each such entity provides a Enable Prejob parameter that affects the symlink existence:

Example

```
[head->wlm[openpbs]]% get enableprejob
yes
```

This parameter is set to yes by cm-wlm-setup when at least one health check is selected as a prejob one. If any health check was configured as a prejob check before cm-wlm-setup execution, and the administrator had a checkmark for that health check, then the prejob is considered enabled.

Workload Manager Configuration For Prolog And Epilog Scripts

The cluster manager configures generic prologs and epilogs during workload manager setup with cm-wlm-setup. The administrator can configure prologs and epilogs using appropriate parameters in the configuration of the workload managers, by creating the symlinks in the local prologs and epilogs directories.

Generic prologs and epilogs are configured by default to run on job compute nodes (one run per each node per job) for Slurm, PBS variants and LSF. However, for UGE the generic prolog runs only on main job node due to the limitations of UGE configuration.

The following parameters for prologs and epilogs can be configured with cmsh or Bright View:

- **Slurm**

  - Prolog Slurmctld: the fully qualified path of a program to execute before granting a new job allocation. The program is executed on the same node where the slurmserver role is assigned. The path corresponds to the PrologSlurmctld parameter in slurm.conf.
  
  - Epilog Slurmctld: the fully qualified path of a program to execute upon termination of a job allocation. The program is executed on the same node where the slurmserver role is assigned. Corresponds with the EpilogSlurmctld parameter in slurm.conf.
- **Prolog**: the fully qualified path of a program to execute on job compute nodes before granting a new job or step allocation. The program corresponds to the Prolog parameter, and by default points to the generic prolog. This prolog runs on every node of the job if the Prolog flags parameter contains the flag Alloc (the default value), otherwise it is executed only on the first node of the job.

- **Epilog**: the fully qualified path of a program to execute on job compute nodes when the job allocation is released.

**LSF**

- **Prolog**: the fully qualified path of a program to execute on the LSF server node on job allocation. As an LSF queue parameter, it corresponds to the PRE_EXEC parameter in lsb.queues.

- **Epilog**: the fully qualified path of a program to execute on the LSF server node on job allocation release. As an LSF queue parameter, it corresponds to the POST_EXEC parameter in lsb.queues.

- **Host prolog**: the fully qualified path of a program to execute on each node of a job before the job is started. Corresponds to the HOST_PRE_EXEC parameter in lsb.queues. By default it is configured to run the generic prolog.

- **Host epilog**: the fully qualified path of a program to execute on each node of a job after the job is finished. Corresponds to the HOST_POST_EXEC parameter in lsb.queues. By default it is configured to run the generic epilog.

**UGE**

- **Prolog**: ugeserver role parameter that points to the fully qualified path of a program to execute. By default it is the generic prolog. It is executed only on the UGE server node.

- **Epilog**: ugeserver role parameter that is empty by default. It can be configured to run the generic epilog. It is executed only on the UGE server node.

**PBS variants**

- **Pelogs**: prolog and epilog hooks that emulate the classic PBS (p)rologue and (p)epilogue scripts located in pbs_mom directory. The pelogs are configured in the appropriate workload manager instance, within pelogs mode when configuring the PBS cluster entity. For example, in cmsh:

  **Example**

  ```
  [bright92]% wlm use openpbs
  [bright92->wlm[openpbs]]% pelogs
  [bright92->wlm[openpbs]]% pelogs% list
  Name (key) Enabled  Order
  ---------- -------- -----
  cm_epilog  yes    99
  cm_prolog  yes    1
  [bright92->wlm[openpbs]]% pelogs% show cm_prolog
  Parameter  Value
  -----------------------------------------------
  Enabled    yes
  Name        cm_prolog
  Events      execjob_begin
  Path        /cm/shared/apps/pbspro/var/cm/cm-pelog-hook.py
  Default action RERUN
  Enable parallel  yes
  ```
7.4 Enabling, Disabling, And Monitoring Workload Managers  

Enabling And Disabling A WLM  
A WLM can be disabled for all nodes with `cm-wlm-setup`. Disabling the WLM means the workload management services are stopped by removing roles, and removing the WLM cluster object.

Alternatively, a WLM can be enabled or disabled by the administrator via role addition and role removal with Bright View or `cmsh`. This is described further on in this section.

Multiple WLM instances of the same type: Versions of NVIDIA Bright Cluster Manager prior to 9.0 already had the ability to have different workload managers run at the same time. However, NVIDIA Bright Cluster Manager version 9.0 introduced the additional ability to run many workload managers of the same kind at the same time.

Example  

```
[bright92-wlm@openpbs->pelogs]% show cm_epilog
```

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>Enabled</td>
<td>yes</td>
</tr>
<tr>
<td>Name</td>
<td>cm_epilog</td>
</tr>
<tr>
<td>Events</td>
<td>execjob_end</td>
</tr>
<tr>
<td>Path</td>
<td>/cm/shared/apps/pbspro/var/cm/cm-pelog-hook.py</td>
</tr>
<tr>
<td>Default action</td>
<td>RERUN</td>
</tr>
<tr>
<td>Enable parallel</td>
<td>yes</td>
</tr>
<tr>
<td>Verbose user output</td>
<td>no</td>
</tr>
<tr>
<td>Torque compatible</td>
<td>no</td>
</tr>
<tr>
<td>Order</td>
<td>99</td>
</tr>
<tr>
<td>Alarm</td>
<td>35</td>
</tr>
<tr>
<td>Debug</td>
<td>no</td>
</tr>
</tbody>
</table>

The parameters of each pelog correspond to appropriate parameters of PBS hooks:

- **Name**: hook name that will be used in PBS
- **Enabled**: flag to enable the hook in PBS
- **Events**: list of PBS events that the hook will run on
- **Path**: path to the hook script that will be imported to PBS in case the hook is not found. The script will not be imported if a hook with the same name already exists in PBS
- **Order**: the hook execution order
- **Alarm**: the hook alarm time (timeout), in seconds

Additional parameters related to the prolog and epilog hooks only are:

- **Default action**: The PBS default action when a prolog or epilog fails
- **Enable parallel**: enable parallel prologues and epilogues, that run on sister moms
- **Verbose user output**: provide verbose hook output to the user’s `.o`/`.e` file
- **Torque compatible**: make torque compatible from prolog/epilog command line arguments point of view

By default, two pelogs are added. These are to run the generic prolog and the generic epilog. If needed, the administrator can add more pelog hooks that will run on different events.
Two WLM instances, Slurm and OpenPBS, are already running at the same time in the cluster, with each WLM assigned to one category. Then, the cluster manager can start up a third WLM instance, such as another Slurm WLM instance. These WLM instances are alternatively called WLM clusters, because they effectively allow one cluster to function as many separate clusters as far as running WLMs is concerned.

From the Bright View or cmsh point of view a WLM consists of

- a WLM server, usually on the head node
- WLM clients, usually on the compute nodes

For the administrator, enabling or disabling the servers or clients is then simply a matter of assigning or unassigning a particular WLM server or client role on the head or compute nodes, as deemed appropriate.

The administrator typically also sets up an appropriate WLM environment module (slurm, uge, openpbs, pbspro, lsf), so that it is loaded up for the end user (section 2.2.3).

### 7.4.1 Enabling And Disabling A WLM With Bright View

A particular WLM package may be installed, but the WLM may not be enabled. This can happen, for example, if disabling a WLM that was previously enabled.

If a WLM instance exists, then the WLM client, submission, and server roles can be enabled or disabled from Bright View by assigning or removing the appropriate roles to nodes, categories, or configuration overlays. Within the role, the properties of the WLM may be further configured by setting options.

**Workload Manager Role Assignment To An Individual Node With Bright View**

**Workload Manager Server**  The following roles are WLM roles that can be assigned to a node:

- server
- submit
- accounting (for the Slurm WLM only, to configure and run the slurmdbd service)
- client

For example, a Slurm server role can be assigned to a head node, bright92, via the clickpath:

```
Devices → Head Nodes[bright92] ↓Edit → Settings → Roles → ADD ↓Slurm server role
```
Options can be set within the workload manager, within the role. For example, for Slurm, a `builtin` or `backfill` option can be set for the `Scheduler` parameter. The workload manager server role is then saved with the selected options (figure 7.7). For starting it up on non-head nodes (but not for a head node), the `imageupdate` command (section 5.6.2) is then run. The workload manager server process and any associated schedulers then automatically start up.

**Workload Manager Client** Similarly, the workload manager client process can be enabled on a node or head node by having the workload manager client role assigned to it. Some basic options can be set for the client role right away.
Saving the role, and then running `imageupdate` (section 5.6.2), automatically starts up the client process with the options chosen, and managed by CMDaemon.

**Workload Manager Role Assignment To A Category With Bright View**

It is true that workload manager role assignment can be done as described in the preceding text for individual non-head nodes. However it is usually more efficient to assign roles using categories or configuration overlays, due to the large number of compute nodes in typical clusters.

Thus, as an example, the case can be considered of all physical on-premises non-head nodes. By default these are placed in the `default` category. This means that, by default, roles in the category are automatically assigned to all those non-head nodes, unless, as an exception, an individual node configuration overrides the category setting and uses a role setting instead at node level.

Viewing the possible workload manager roles for the category `default` is done by using the click-path:

```
Grouping → Categories[default] → Edit → Settings → Roles → Add
```

Once the role is selected, its options can be edited and saved.

For compute nodes, the role assigned is usually a workload manager client. If the assigned role is that of a workload manager client, then the node with that role can have queues, GPUs, and other parameters specified for it.

For example, queues can then be assigned via the clickpath:

```
HPC → WLM Management Clusters[cluster instance] → Job Queues
```

while GPUs, if using Slurm as the workload manager with default settings, can then be specified via the clickpath:

```
Configuration Overlays → slurm-client-gpu → roles → slurm-client ↓ edit → Generic Resources → gpu
```

The workload manager server role can also be assigned to a non-head node. For example, a Slurm server role can be taken on by a non-head node. This is the equivalent to the `--server-nodes` option of `cm-wlm-setup`.

Saving the roles with their options and then running `imageupdate` (section 5.6.2) automatically starts up the newly-configured workload manager.

**Workload Manager Role Options With Bright View**

Each compute node role (workload manager client role) has options that can be set for GPUs, Queues, and Slots.

- **Slots**, in a workload manager, corresponds in the cluster manager to:
  - the `CPUs` setting (a `NodeName` parameter) in Slurm’s `slurm.conf`
  - the `np` setting in PBS,

  and is normally set to the number of cores per node.

  In LSF setting the number of slots for the client role to 0 means that the client node does not run jobs on itself, but becomes a submit host, which means it is able to forward jobs to other client nodes.

- **Queues** with a specified name are available in their associated role after they are created. The creation of queues is described in sections 7.6.2 (using Bright View) and 7.7.2 (using `cmsh`).

  All server roles also provide the option to enable or disable the `External Server` setting. Enabling that means that the server is no longer managed by the cluster manager, but provided by an external device.

  The `cmsh` equivalent of enabling an external server is described on page 309.
7.4 Enabling, Disabling, And Monitoring Workload Managers

7.4.2 Enabling And Disabling A Workload Manager With cmsh

A particular workload manager package may be set up, but not enabled. This can happen, for example, if no WLM server or WLM client role has been assigned.

If a WLM instance exists, then the WLM client, server, or submit roles can be enabled from cmsh by assigning it from within the roles submode. Within the assigned role, the properties of the WLM may be further configured by setting options.

Workload Manager Role Assignment To A Configuration Overlay With cmsh

In cmsh, workload manager role assignment to a configuration overlay (section 2.1.5) can be done using configurationoverlay mode. By default cm-wlm-setup run as an Ncurses session creates some configuration overlays with suggestive names, and assigns roles to the configuration overlays according to what the names suggest. Thus, for example, with the cm-wlm-setup Ncurses session used to carry out an express setup for Slurm, the configuration overlays that get created are the following:

Example

```
[bright92->configurationoverlay]% list
Name (key)          Priority All head nodes Nodes Categories Roles
-------------------- ---------- -------------- ---------------- ---------------- ----------------
slurm-accounting    500       yes slurmaccounting
slurm-client        500       no  default slurmclient
slurm-server        500       yes slurmserver
slurm-submit        500       no  default slurmsubmit
wlm-headnode-submit 600       yes slurmsubmit
```

Nodes in the default category can take on the slurmclient or slurmsubmit role by setting the nodes for the role using the associated configuration overlays slurm-client or slurmsubmit.

The wlm-headnode-submit configuration overlay is a special overlay. It is applied only to the head node, and is shared among all installed workload managers. Setting this overlay means that the head node, by default, has a submit role for a given workload manager.

Example

```
[bright92->configurationoverlay]% use slurm-client
[bright92->configurationoverlay[slurm-client]]% show
Parameter Value
------------------------------------------------------------------------------
Name                   slurm-client
Revision   
All head nodes no
Priority  500
Nodes
Categories  default
Roles      slurmclient
Customizations <0 in submode>
[bright92->configurationoverlay[slurm-client]]% set nodes node001 node002 node003 bright92
[bright92->configurationoverlay[slurm-client]]% set nodes node001..node002
[bright92->configurationoverlay[slurm-client]]% commit
```

```
[bright92->configurationoverlay[slurm-client]]% list
Name (key)          Priority All head nodes Nodes Categories Roles
-------------------- ---------- -------------- ---------------- ---------------- ----------------
slurm-accounting    500       yes slurmaccounting
slurm-client        500       no  node001,node002 default slurmclient
slurm-server        500       yes slurmserver
```
slurm-submit 500 no default slurm-submit
wlm-headnode-submit 600 yes slurmsubmit

All the head nodes can also be made to take on the configuration overlay role by setting its All head nodes value to yes. The union set of All head nodes with Nodes is the set of nodes to which the role is applied for that configuration overlay.

Values for the parameters in a role, such as the slurmclient role, can be set within the configuration overlay:

Example

```
[slurm-client]
roles
[slurm-client] use slurmclient
[slurm-client] show
```

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>Name</td>
<td>slurmclient</td>
</tr>
<tr>
<td>Revision</td>
<td></td>
</tr>
<tr>
<td>Type</td>
<td>SlurmClientRole</td>
</tr>
<tr>
<td>Add services</td>
<td>yes</td>
</tr>
<tr>
<td>WLM cluster</td>
<td>slurm</td>
</tr>
<tr>
<td>Slots</td>
<td>0</td>
</tr>
<tr>
<td>All Queues</td>
<td>no</td>
</tr>
<tr>
<td>Queues</td>
<td>defq</td>
</tr>
<tr>
<td>Provisioning associations</td>
<td>&lt;0 internally used&gt;</td>
</tr>
<tr>
<td>Power Saving Allowed</td>
<td>no</td>
</tr>
<tr>
<td>Features</td>
<td></td>
</tr>
<tr>
<td>Sockets</td>
<td>0</td>
</tr>
<tr>
<td>Cores Per Socket</td>
<td>0</td>
</tr>
<tr>
<td>ThreadsPerCore</td>
<td>0</td>
</tr>
<tr>
<td>Boards</td>
<td>0</td>
</tr>
<tr>
<td>SocketsPerBoard</td>
<td>0</td>
</tr>
<tr>
<td>RealMemory</td>
<td>0</td>
</tr>
<tr>
<td>NodeAddr</td>
<td>0</td>
</tr>
<tr>
<td>Weight</td>
<td>0</td>
</tr>
<tr>
<td>Port</td>
<td>0</td>
</tr>
<tr>
<td>TmpDisk</td>
<td>0</td>
</tr>
<tr>
<td>Reason</td>
<td></td>
</tr>
<tr>
<td>CPU Spec List</td>
<td>0</td>
</tr>
<tr>
<td>Core Spec Count</td>
<td>0</td>
</tr>
<tr>
<td>Mem Spec Limit</td>
<td>0</td>
</tr>
<tr>
<td>Node Customizations</td>
<td>&lt;0 in submode&gt;</td>
</tr>
<tr>
<td>Generic Resources</td>
<td>&lt;0 in submode&gt;</td>
</tr>
</tbody>
</table>

After the workload manager roles are assigned or unassigned, and after running imageupdate (section 5.6.2) for non-head nodes, the associated workload manager services automatically start up or stop as appropriate.

The configuration overlay role values are inherited by categories and nodes, unless the categories and nodes have their own values set. Thus, for role properties, a value set at node level overrides values set at category level, and a value set at configuration overlay level overrides a value set at category level. This is typical of how properties of objects are inherited in the cluster manager levels.
Workload Manager Role Assignment To A Category With cmsh

In cmsh, workload manager role assignment to a node category can be done using category mode, using the category name, assigning a role from the roles submode, setting the WLM instance for that role, and committing the modified role:

Example

```
[root@bright92 ~]# cmsh
[bright92]# category
[bright92->category]# use default
[bright92->category[default]]# roles
[bright92->category[default]->roles]# assign slurmclient
[bright92->category[default]->roles*[slurmclient*]]# wlm list
```

<table>
<thead>
<tr>
<th>Type</th>
<th>Name (key)</th>
<th>Server nodes</th>
<th>Submit nodes</th>
<th>Client nodes</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>slurm</td>
<td>bright92</td>
<td>bright92, node001 node001, node002</td>
<td></td>
</tr>
</tbody>
</table>

```
[bright92->category[default]->roles*[slurmclient*]]# set wlmcluster slurm1
[bright92->category[default]->roles*[slurmclient*]]# commit
```

Settings that are assigned in the slurmclient role of the category overrule the slurmclient role configuration overlay settings.

The role assignment at category level requires the value for a WLM instance to be specified for wlmcluster before the commit command is successful.

Workload Manager Role Assignment To An Individual Node With cmsh

In cmsh, assigning a workload manager role to a head node can be done in device mode. This can be done by using the head node name as the device, assigning the workload manager role to the device, setting the WLM instance value to the role within the role submode, and committing the modified role.

Example

```
[root@bright92 ~]# cmsh
[bright92]# device
[bright92->device]# use bright92
[bright92->device[bright92]]# roles
[bright92->device[bright92]->roles]# assign slurmserver
[bright92->device[bright92]->roles*[slurmserver*]]# wlm list
```

<table>
<thead>
<tr>
<th>Type</th>
<th>Name (key)</th>
<th>Server nodes</th>
<th>Submit nodes</th>
<th>Client nodes</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>slurm</td>
<td>bright92</td>
<td>bright92, node001 node001, node002</td>
<td></td>
</tr>
</tbody>
</table>

```
[bright92->device[bright92]->roles*[slurmserver*]]# set wlmcluster slurm1
[bright92->device[bright92]->roles*[slurmserver*]]# commit
```

For regular nodes, role assignment is done via device mode, using the node name. The node name is assigned the workload manager role, the WLM instance value is set for that role in the role submode, and the modified role is committed.

Example

```
[root@bright92 ~]# cmsh
[bright92]# device
[bright92->device]# use node001
[bright92->device[node001]]# roles
[bright92->device[node001]->roles]# assign slurmclient
[bright92->device[node001]->roles*[slurmclient*]]# wlm list
```
The role assignment at node level requires the value for a WLM instance to be specified for \texttt{wlmcluster} before the \texttt{commit} command is successful.

Role assignment values set in \texttt{device} mode have precedence over any role assignment values set in \texttt{category} mode for that node. This means, for example, that if a node is originally in a node category with a \texttt{slurmclient} role and queues set, then when the node is assigned a \texttt{slurmclient} role from \texttt{device} mode, its queue properties are empty by default.

**Setting Options For Workload Manager Settings With cmsh**

In the preceding text, it is explained how the workload manager client or server is assigned a role (such as \texttt{slurmclient} or \texttt{slurmserver}) within the \texttt{roles} submode. It is done from within a main mode of \texttt{cmsh}. The main modes from which role assignment can be done are: \texttt{configurationoverlay}, \texttt{category} or \texttt{device}.

**Options for workload managers in general:** Whatever main mode is used, the workload manager options for a role can then be set with the usual object commands introduced in section 2.5.3.

- **WLM client options:** For example, the configuration options of a WLM client, such as the PBS Pro client, can be seen by using the \texttt{show} command on the role. Here it can be seen at a \texttt{category} level, for the default category \texttt{default}:

**Example**

```
[bright92->category[default]->roles[pbsproclient]]% show

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>Add services</td>
<td>yes</td>
</tr>
<tr>
<td>All Queues</td>
<td>no</td>
</tr>
<tr>
<td>GPUs</td>
<td>0</td>
</tr>
<tr>
<td>Name</td>
<td>pbsproclient</td>
</tr>
<tr>
<td>Properties</td>
<td></td>
</tr>
<tr>
<td>Provisioning associations</td>
<td>&lt;0 internally used&gt;</td>
</tr>
<tr>
<td>Queues</td>
<td></td>
</tr>
<tr>
<td>Revision</td>
<td></td>
</tr>
<tr>
<td>Slots</td>
<td>1</td>
</tr>
<tr>
<td>Type</td>
<td>PbsProClientRole</td>
</tr>
<tr>
<td>WLM cluster</td>
<td></td>
</tr>
<tr>
<td>Mom Settings</td>
<td>&lt;submode&gt;</td>
</tr>
<tr>
<td>Comm Settings</td>
<td>&lt;submode&gt;</td>
</tr>
<tr>
<td>Node Customizations</td>
<td>&lt;0 in submode&gt;</td>
</tr>
</tbody>
</table>
```

The \texttt{Slots} option can be set in the role:

**Example**

```
[bright92->category[default]->roles[pbsproclient]]% set slots 2
[bright92->category[default]->roles[pbsproclient]]% commit
```
• **WLM server options:** Similarly, WLM server options can be managed from an assigned server role. For PBS, the `pbsproserver` role for a device shows:

**Example**

```
[bright92->device[bright92]->roles]% use pbsproserver
[bright92->device[bright92]->roles[pbsproserver]]% show
```

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>Name</td>
<td>pbsproserver</td>
</tr>
<tr>
<td>Revision</td>
<td></td>
</tr>
<tr>
<td>Type</td>
<td>PbsProServerRole</td>
</tr>
<tr>
<td>Add services</td>
<td>yes</td>
</tr>
<tr>
<td>WLM cluster</td>
<td></td>
</tr>
<tr>
<td>Provisioning associations</td>
<td>&lt;0 internally used&gt;</td>
</tr>
<tr>
<td>External Server</td>
<td>no</td>
</tr>
<tr>
<td>Comm Settings</td>
<td>&lt;submode&gt;</td>
</tr>
</tbody>
</table>

**Option to set an external workload manager:** A workload manager can be set to run as an external server from within a device mode role:

**Example**

```
[bright92->device[bright92]->roles[pbsproserver]]% set externalserver on
[bright92->device[bright92]->roles[pbsproserver*]]% commit
```

For convenience, setting it on the head node is recommended.

The Bright View equivalent of configuring `externalserver` is described on page 304.

### 7.4.3 Monitoring The Workload Manager Services

By default, the workload manager services are monitored. The cluster manager attempts to restart the services using the service tools (section 3.11), unless the role for that workload manager service is disabled, or the service has been stopped.

Workload manager roles and corresponding services can be disabled using `cm-wlm-setup` (section 7.3), Bright View role configuration (section 7.4.1), or `cmsh` role configuration (section 7.4.2).

The daemon service states can be viewed for each node via the shell, `cmsh`, or Bright View (section 3.11).

Queue submission and scheduling daemons normally run on the head node. From Bright View their states are viewable via the clickpath to the services running on the node. For example, on a head node (figure 7.8), via the clickpath:

```
Devices→Head Nodes→[bright92]→Settings→Services
```
For a regular node, a similar clickpath for node001, for example, is:
Devices → Nodes → node001 → Settings → Services
and leads to a view of services on the regular nodes (figure 7.9)

Considering only the WLMs: in figure 7.8 the pbsserver is seen running on the head node, while in figure 7.9 the pbsmom server is seen running on the compute node.

The roles that set up that result in those servers running on the head node and compute nodes are seen in figures 7.10 and 7.11:
7.4 Enabling, Disabling, And Monitoring Workload Managers

The roles seen in these figures are from the defaults that \texttt{cm-wlm-setup} provides in an express setup.

For regular nodes, the inheritance of roles from category level or configuration overlay level is indicated by the values in the \texttt{INHERITED} column. Thus, in figure 7.11, the \texttt{pbsprosubmit} role is decided by the default setting from the category level, while the \texttt{pbsproclient} role is decided by the default setting in the \texttt{openpbs-client} overlay.

The assignment of roles can be varied to taste for WLMs. This allows WLM services to run on the head node or on the regular nodes.

From \texttt{cmsh} the services states are viewable from within \texttt{device} mode, using the \texttt{services} command. One-liners from the shell to illustrate this are (output elided):

\textbf{Example}

```
[root@bright92 ~]# cmsh -c "device services node001; status"
Service Status
---------- ---------
msl0d [ UP ]
pbsmom [ UP ]
[root@bright92 ~]# cmsh -c "device services bright92; status"
Service Status
---------- ---------

```
Roles can be viewed from within the main modes of configuration, category, or device. One-liners to view these are:

Example

```
[root@bright92 ~]# cmsh -c "configurationoverlay; list"
Name (key)      Priority All head nodes  Nodes  Categories  Roles
--------------- ---------- -------------- ---------------- ---------------- ----------------
opembs-client   500       no          default     pbsproclient
opembs-server   500       yes     pbsproserver
opembs-submit   500       no          default     pbsprosubmit
[overlay:openpbs-client:500] pbsproclient
[overlay:openpbs-submit:500] pbsprosubmit
[overlay:openpbs-client:500] pbsproclient
[overlay:openpbs-submit:500] pbsprosubmit
```

The -p|--priority option displays of the list command the priority setting for the roles.

7.5 Configuring And Running Individual Workload Managers

The cluster manager deals with the various choices of workload managers in as generic a way as possible. This means that not all features of a particular workload manager can be controlled, so that fine-tuning must be done through the workload manager configuration files. Workload manager configuration files that are controlled by the cluster manager should normally not be changed directly because the cluster manager overwrites them. However, overwriting by CMDaemon is prevented on setting the directive:

```
FreezeChangesTo<workload manager>Config = <true|false>
```
in `cmd.conf` (Appendix C), where `<workload manager>` takes the value of Slurm, UGE, LSF, or PBSPro, as appropriate. The value of the directive defaults to `false`.

A list of configuration files that are changed by CMDaemon, the items changed, and the events causing such a change are listed in Appendix H.

A very short guide to some specific workload manager commands that can be used outside of the NVIDIA Bright Cluster Manager 9.2 system is given in Appendix F.

### 7.5.1 Configuring And Running Slurm

#### Slurm Version

At the time of writing (April 2022), the cluster manager is integrated with Slurm for the following Slurm packages from the cluster manager repositories:

- `slurm20`, `slurm20-*`: For Slurm version 20.02.7
- `slurm20.11`, `slurm20.11-*`: For Slurm version 20.11.8
- `slurm21.08`, `slurm21.08-*`: For Slurm version 21.08.5

The distribution version of Slurm (package: `slurm`) is not integrated with the cluster manager and conflicts with the preceding packages. It should not be used.

The exact list of packages can be found by running, for example:

```bash
[root@bright92 ~]# i="slurm20 slurm20.11 slurm21.08"
  for j in $i ; do echo "===="; yum info $j $j-* | grep ^Name; done
====
Name : slurm20
Name : slurm20-client
Name : slurm20-contribs
Name : slurm20-devel
Name : slurm20-example-configs
Name : slurm20-openlava
Name : slurm20-pam
Name : slurm20-perlapi
Name : slurm20-slurmdbd
Name : slurm20-slurmrestd
Name : slurm20-torque
====
Name : slurm20.11
Name : slurm20.11-client
Name : slurm20.11-contribs
```

Important updates from upstream are patched into the cluster manager repositories.

**Updating From** `slurm20.11` **To** `slurm21.08`

Upgrading between major versions of Slurm is possible. It is a good idea to upgrade one version at a time, rather than jumping 2 or more versions ahead, which requires a full wipe of the Slurm configuration.

An upgrade from one major version of Slurm to another can be carried out according to the following example, which is for an update from major version 20.11 to version 21.08, and avoids total reconfiguration of the Slurm configuration:

- It is recommended that no jobs are running. Draining nodes (section 7.7.3) is one way to arrange this over time. No new jobs run on a drained node, but old ones are allowed to finish.
• When all running jobs are finished, then Slurm server services—slurmctld and slurmdbd—should be stopped using cmsh or Bright View (section 3.11.2):

Example

[bright92->device[bright92]->services] % stop slurmctld
[bright92->device[bright92]->services] % stop slurmdbd

• The old Slurm packages should then be removed. There can be only one version of Slurm at a time, so there will be a package installation conflict if a new version is installed while an old one is still there.

Removal can be carried out on RHEL-based systems with, for example:

[root@bright92 ~] # yum remove slurm20.11*

The old packages must also be removed from each software image that uses it:

[root@bright92 ~] # cm-chroot-sw-img /cm/images/<software image>
...
[root@<software image> /] # yum remove slurm20.11*  
...removal takes place ...
[root@<software image> /] # exit

• The new packages can then be installed. For installation onto the RHEL head node, the installation might be carried out as follows:

[root@bright92 ~] # yum install slurm21.08 slurm21.08-client slurm21.08-contribs slurm21.08-slurmdbd \ slurm21.08-perlapi slurm21.08-devel

The client package can be installed in each software image with, for example:

[root@bright92 ~] # cm-chroot-sw-img /cm/images/<software image>
...
[root@<software image> /] # yum install slurm21.08-client  
...installation takes place
[root@<software image> /] # exit

Other Slurm packages from the repository may also be installed on the head node and within the software images, as needed.

• The new Slurm version is then set in cmsh or Bright View, in the Slurm WLM cluster configuration:

Example

[root@bright92 ~] # cmsh
[bright92]% wlm use slurm
[bright92->wlm[slurm]]% set version 21.08; commit

• Slurm server services slurmctld and slurmdbd should then be started again using cmsh or Bright View:

Example
7.5 Configuring And Running Individual Workload Managers

The nodes can then have their new image placed on them, and the new Slurm configuration can then be taken up. This can be done in the following two ways:

1. The regular nodes can then be restarted to supply the live nodes with the new image and get the new Slurm configuration running.

2. Alternatively, the imageupdate command (section 5.6.2) can be run on the live nodes to supply them with the image.

Running the imageupdate command in dry mode (the default) first is recommended. The synclog command can then be run to check there are no unexpected changes that will take place due to the update. If all is well, then imageupdate's wet mode flag `-w` can be used in order to really carry out the task.

For example, the change can be checked, and then actually carried out, for the image on node001 with:

```
[root@bright92 ~]# cmsh
[bright92]device use node001
[bright92->device[node001]]% imageupdate
Performing dry run (use synclog command to review result, then pass -w to perform real update)...
...some messages...
imageupdate [ COMPLETED ]
[bright92->device[node001]]% synclog
...rsync dry run output...
[bright92->device[node001]]% imageupdate -w
...same messages as before, but this time it really happens...
[bright92->device[node001]]% commit
```

The last commit command triggers the reconfiguration of the file `/etc/systemd/system/slurmd.service.d/99-cmd.conf` on the node. After a short time—around 30 seconds—the file is regenerated. The slurmd service on the node can then be restarted with:

```
[root@bright92 ~]# ssh node001 "systemctl daemon-reload"
```

### Configuring Slurm

After package setup is done with `cm-wlm-setup` (section 7.3), Slurm software components are installed in a symlinked directory `/cm/shared/apps/slurm/current`.

Slurm clients and servers can be configured to some extent via role assignment (sections 7.4.1 and 7.4.2).

Using `cmsh`, advanced option parameters can be set under the `slurmclient` and `slurmserver` roles. The settings for the roles can be done at configuration overlay, category, or node level (sections 2.1.5, 2.1.6).

By default, the `cm-wlm-setup` utility configures Slurm using configuration overlays.

### Example

```
[bright92->configurationoverlay]% list
Name (key)   Priority All head nodes Nodes Categories Roles
-----------  -------- --------------- -------------- ---------- ----------
slurm-accounting  500    yes             slurmaaccounting
slurm-client  500      no default    slurmclient
slurm-server  500      no             slurmserver
slurm-submit  500      no             slurmsubmit
vlm-headnode-submit 600    yes
```
The settings within the roles can be viewed and modified. For example, the slurmclient role of the slurm-client configuration overlay can be viewed:

Example

[bright92->configurationoverlay]% roles slurm-client
[bright92->configurationoverlay[slurm-client]->roles]% show

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>Name</td>
<td>slurmclient</td>
</tr>
<tr>
<td>Revision</td>
<td>slurmclient</td>
</tr>
<tr>
<td>Type</td>
<td>SlurmClientRole</td>
</tr>
<tr>
<td>Add services</td>
<td>yes</td>
</tr>
<tr>
<td>WLM cluster</td>
<td>slurm</td>
</tr>
<tr>
<td>Slots</td>
<td>0</td>
</tr>
<tr>
<td>All Queues</td>
<td>no</td>
</tr>
<tr>
<td>Queues</td>
<td>defq</td>
</tr>
<tr>
<td>Provisioning associations</td>
<td>&lt;0 internally used</td>
</tr>
<tr>
<td>Features</td>
<td></td>
</tr>
<tr>
<td>Sockets</td>
<td>0</td>
</tr>
<tr>
<td>Cores Per Socket</td>
<td>0</td>
</tr>
<tr>
<td>ThreadsPerCore</td>
<td>0</td>
</tr>
<tr>
<td>Boards</td>
<td>0</td>
</tr>
<tr>
<td>SocketsPerBoard</td>
<td>0</td>
</tr>
<tr>
<td>RealMemory</td>
<td>0B</td>
</tr>
<tr>
<td>NodeAddr</td>
<td></td>
</tr>
<tr>
<td>Weight</td>
<td>0</td>
</tr>
<tr>
<td>Port</td>
<td>0</td>
</tr>
<tr>
<td>TmpDisk</td>
<td>0</td>
</tr>
<tr>
<td>Reason</td>
<td></td>
</tr>
<tr>
<td>CPU Spec List</td>
<td></td>
</tr>
<tr>
<td>Core Spec Count</td>
<td>2</td>
</tr>
<tr>
<td>Mem Spec Limit</td>
<td>0B</td>
</tr>
<tr>
<td>Auto Detect</td>
<td>None</td>
</tr>
<tr>
<td>Node Customizations</td>
<td>&lt;0 in submode</td>
</tr>
<tr>
<td>Generic Resources</td>
<td>&lt;0 in submode</td>
</tr>
<tr>
<td>Cpu Bindings</td>
<td>None</td>
</tr>
</tbody>
</table>

The settings can be modified. For example Core Spec Count:

Example

[bright92->configurationoverlay[slurm-client]->roles]% set slurmclient corespeccount 2
[bright92->configurationoverlay[slurm-client]->roles]% commit

As usual, values set at node level override the values set at categories and configuration overlays level.

For example, to set corespeccount to 4, only for node001 but not for other nodes, the session might run further as:

Example

[bright92->configurationoverlay[slurm-client]->roles]% device use node001
[bright92->device[node001]]% roles
[bright92->device[node001]]% assign slurmclient
[bright92->device[node001]]% set corespeccount 4
[bright92->device[node001]]% assignslurmclient
[bright92->device[node001]]% device[slurmclient]* roles*
[bright92->device[node001]]% roles*
[bright92->device[node001]]% set corespeccount 4
[bright92->device[node001]]% commit
In the preceding session, the role needs to be assigned at node level with assign slurmclient because it does not initially exist at node level. If it already existed, then use slurmclient could have been used to descend into that role.

Also in the preceding session, one of the values that the Slurm client needs to know is wlmcluster, which decides which WLM it is to work with on the cluster. The value is selected from the list of WLM instance names in wlm mode.

The level of the active role can be seen with the list command. For example, the Slurm client role assignment at node level is seen here:

Example

```
[bright92->device[node001]->roles[slurmclient]]% list
Name (key) -----------------------------------
[overlay:slurm-submit] slurmsubmit
[overlay:slurm-client] slurmclient
```

Removing the assignment has the list command display the configuration overlay Slurm client role assignment:

Example

```
[bright92->device[node001]->roles[slurmclient]]% unassign slurmclient; commit
[bright92->device[node001]->roles[slurmclient]]% list
Name (key) -----------------------------------
[overlay:slurm-submit] slurmsubmit
[overlay:slurm-client] slurmclient
```

Generic resources (gres) configuration in Slurm: In order to configure generic resources, the genericresources mode can be used to set a list of objects. Each object then represents one generic resource available on nodes.

Each value of name in genericresources must already be defined in the list of GresTypes. The list of GresTypes is defined in the slurmserver role.

Several generic resources entries can have the same value for name (for example gpu), but must have a unique alias. The alias is a string that is used to manage the resource entry in cmsh or in Bright View. The string is enclosed in square brackets in cmsh, and is used instead of the name for the object. The alias does not affect Slurm configuration.

For example, to add two GPUs for all the nodes in the default category which are of type k20xm, and to assign them to different CPU cores, the following cmsh commands can be run:

Example

```
[bright92]% category use default
[bright92->category[default]]% roles
```
In Bright View, the clickpath:

Configuration Overlays → slurm-client-gpu → Edit → Roles → slurmclient → Edit → Generic Resources → ADD

provides the equivalent (figure 7.12):

![Slurm Generic Resource](image)

Figure 7.12: Bright View access to NVIDIA GPU configuration options

After the generic resources are committed, the cluster manager updates the `gres.conf` file.

Since NVIDIA Bright Cluster Manager version 8.2 and higher, a single `gres.conf` configuration file, located at `/cm/shared/apps/slurm/var/etc/slurm/gres.conf` is used.

If the category consists of node001 and node002, then the entries to the `gres.conf` file in this case would look like:
GPU autodetection: Slurm supports GPU autodetection from version 20 onwards, using the `AutoDetect` flag in `gres.conf`.

NVIDIA Bright Cluster Manager 9.0 and onward provide a `slurm20` package built with GPU autodetection enabled. Both AMD and NVidia GPUs can then be detected if autodetection is also enabled in `gres.conf`.

NVIDIA Bright Cluster Manager 9.1 onwards allows the `AutoDetect` flag to be set via `cmsh` or Bright View.

There are 3 options for the flag:

1. `None`: do not autodetect any GPUs (`AutoDetect` is not added to `gres.conf`),
2. `NVML`: detect NVidia GPUs (`AutoDetect=nvml` is added to `gres.conf`),
3. `RSMI`: detect AMD GPUs (`AutoDetect=rsmi` is added to `gres.conf`).

The default value is `None`.

GPU autodetection removes the need for explicit GPU configuration in `gres.conf`, although the `Gres=...` line in `slurm.conf` is still required in order to tell `slurmctld` how many GRES to expect. This means the administrator needs to define GPUs in the `genericresources` mode of the `slurmclient` role.

However there is no need in this case to specify all the details. It is enough to add a single generic resource named `gpu`, and to specify the number of such GPUs. In order to skip adding the generic resources, the flag `AddToGresConfig` should be set in the `genericresource` entity. For example:

Example

```
[bright92] configurationoverlay use slurm-client
[bright92->configurationoverlay[slurm-client]] roles
[bright92->configurationoverlay[slurm-client]->roles] use slurmclient
....[slurmclient] genericresources
....[slurmclient]->genericresources% add autodetected-gpus
....[slurmclient]->genericresources*[autodetected-gpus*] set name gpu
....[slurmclient]->genericresources*[autodetected-gpus*] set count 8
....[slurmclient]->genericresources*[autodetected-gpus*] set addtogroupconfig no
....[slurmclient]->genericresources*[autodetected-gpus*] set commit
....[slurmclient]->genericresources*[autodetected-gpus*] set wlm use slurm
[bright92->wlm[slurm]] set autodetect nvml
[bright92->wlm*[slurm*]] set commit
[bright92->wlm[slurm]]
```

Slurm GPU cluster settings: When a GPU is configured the following parameters can be configured with the cluster manager:
Workload Management

<table>
<thead>
<tr>
<th>Bright View Option</th>
<th>Slurm Object Value Within cmsh</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Select Type</td>
<td>SelectType</td>
<td>Identifies the type of resource selection algorithm to be used. Changing this value causes the <code>slurmctld</code> daemon to be restarted. Default: Consumable trackable resource: <code>select/cons_tres</code></td>
</tr>
<tr>
<td>Select Type Parameters</td>
<td>SelectTypeParameters</td>
<td>Parameters for the resource selection algorithm. Acceptable values for these parameters depend upon the algorithm selected. Default: Core as a consumable resource: <code>Cr_Core</code></td>
</tr>
<tr>
<td>Accounting Storage TRES</td>
<td>AccountingStorageTRES</td>
<td>Comma-separated list of resources to be tracked on the cluster, requested by an <code>sbatch</code>/<code>srun</code> job on submission. Default: GPU as a generic resource: <code>gres/gpu</code></td>
</tr>
</tbody>
</table>

Example

```
[bright92->wlm[slurm]]% get selecttype
select/cons_tres
```

**Slurm node settings:** The parameter value for a Slurm option in `slurm.conf` is set by CMDaemon, if its value is not: 0.

A parameter value of 0 means that the default values of Slurm are used. These usually have the value: 0.

The advanced options that CMDaemon manages for Slurm are:

<table>
<thead>
<tr>
<th>Bright View Option</th>
<th>Slurm Option</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Features</td>
<td>Feature=&lt;string&gt; entry in the file <code>slurm.conf</code></td>
<td>Arbitrary strings can be entered to indicate some characteristics of a node, one string per entry. For example: <code>text1</code>, <code>text2</code> and so on. These become part of the: <code>Feature=text1,text2...</code> attribute to the <code>NodeName=&lt;node name&gt;</code> entry line in <code>slurm.conf</code>, as indicated in <code>man(5) slurm.conf</code>. The strings also become added attributes to the <code>GresTypes</code> entry of that file. Default: blank.</td>
</tr>
</tbody>
</table>

...continues
## 7.5 Configuring And Running Individual Workload Managers

...continued

<table>
<thead>
<tr>
<th>Bright View Option</th>
<th>Slurm Option</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Slots</td>
<td>CPU</td>
<td>Number of logical processors on the node. For Slurm 20 and beyond, CMDaemon detects the number of CPU cores, and sets <code>procs=number of cores</code> via slots autodetection. Default for Slurm prior to version 20: 0</td>
</tr>
<tr>
<td>Sockets</td>
<td>Sockets</td>
<td>Processor chips on node. If this is defined, then <code>SocketsPerBoard</code> must not be defined. Default: 0</td>
</tr>
<tr>
<td>Cores per socket</td>
<td>CoresPerSocket</td>
<td>Number of cores per socket. Default: 0</td>
</tr>
<tr>
<td>ThreadsPerCore</td>
<td>ThreadsPerCore</td>
<td>Number of logical threads for a single core. Default: 0</td>
</tr>
<tr>
<td>Boards</td>
<td>Boards</td>
<td>Number of baseboards in a node. Default: 0</td>
</tr>
<tr>
<td>SocketsPerBoard</td>
<td>SocketsPerBoard</td>
<td>Number of processor chips on baseboard. If this is defined, then <code>Sockets</code> must not be defined. Default: 0</td>
</tr>
<tr>
<td>RealMemory</td>
<td>RealMemory</td>
<td>Size of real memory on the node, MB. Default: 0</td>
</tr>
<tr>
<td>NodeHostname</td>
<td>NodeHostname</td>
<td>Default: as defined by Slurm’s <code>NodeName</code> parameter.</td>
</tr>
<tr>
<td>NodeAddr</td>
<td>NodeAddr</td>
<td>Default: as set by Slurm’s <code>NodeHostname</code> parameter.</td>
</tr>
<tr>
<td>State</td>
<td>State</td>
<td>State of the node with user jobs. Possible Slurm values are: <code>DOWN</code>, <code>DRAIN</code>, <code>FAIL</code>, <code>FAILING</code>, and <code>UNKNOWN</code>. Default: <code>UNKNOWN</code></td>
</tr>
<tr>
<td>Weight</td>
<td>Weight</td>
<td>The priority of the node for scheduling. Default: 0</td>
</tr>
</tbody>
</table>

...continues
### Workload Management

...continued

<table>
<thead>
<tr>
<th><strong>Bright View Option</strong></th>
<th><strong>Slurm Option</strong></th>
<th><strong>Description</strong></th>
</tr>
</thead>
<tbody>
<tr>
<td>Port</td>
<td>Port</td>
<td>Port that slurmd listens to on the compute node. Default: as defined by SlurmPort parameter. If SlurmPort is not specified during build: Default: 6818.</td>
</tr>
<tr>
<td>TmpDisk</td>
<td>TmpDisk</td>
<td>Total size of Slurm’s temporary filesystem, TmpFS, typically /tmp, in MB. TmpFS is the storage location available to user jobs for temporary storage. Default: 0</td>
</tr>
<tr>
<td>Options</td>
<td>extra options</td>
<td>Extra options that are added to slurm.conf</td>
</tr>
</tbody>
</table>

Further Slurm documentation is available:

- via man pages under /cm/shared/apps/slurm/current/man.
- as HTML documentation in the directory /cm/shared/apps/slurm/current/share/doc/slurm-21.08.5/html
- at the Slurm website at http://slurm.schedmd.com/documentation.html

Slurm is set up with reasonable defaults, but administrators familiar with Slurm can reconfigure the configuration file using a web browser with the following JavaScript-based configuration generators:

- /cm/shared/apps/slurm/current/share/doc/slurm-21.08.5/html/configurator.easy.html: this is a simplified configurator
- /cm/shared/apps/slurm/current/share/doc/slurm-21.08.5/html/configurator.html: this is the full version of the configurator.

If the configuration file becomes mangled beyond repair, the original default can be regenerated once again by re-installing the Slurm package, then running the script /cm/shared/apps/slurm/var/cm/cm-restore-db-password, and then running cm-wlm-setup. Care must be taken to avoid duplicate parameters being set in the configuration file—slurmd may not function correctly in such a configuration.

### Running Slurm

Slurm can be disabled and re-initialized with the cm-wlm-setup tool (section 7.3) during package installation itself.

Alternatively, role assignment and role removal can be used to adjust what nodes, if any, run Slurm. The assignment and removal of roles can be carried out from Bright View (section 7.4.1) or cmsh (section 7.4.2).

The Slurm workload manager runs these daemons:

1. as servers:
   - slurmd: The database that tracks job accounting. It is part of the slurmd service.
   - slurmctld: The controller daemon. Monitors Slurm processes, accepts jobs, and assigns resources. It is part of the slurm service.
   - munged: The authentication (client-and-server) daemon. It is part of the munge service.

2. as clients:
7.5 Configuring And Running Individual Workload Managers

(a) **slurmd**: The compute node daemon that monitors and handles tasks allocated by **slurmctld** to the node. It is part of the **slurm** service.

(b) **slurmstepd**: A temporary process spawned by the **slurmd** compute node daemon to handle Slurm job steps. It is not initiated directly by users or administrators.

(c) **munged**: The authentication (client-and-server) daemon. It is part of the **munge** service.

Logs for the daemons are saved on the node that they run on. Accordingly, the locations are:

- `/var/log/slurmdbd`
- `/var/log/slurmd`
- `/var/log/slurmctld`
- `/var/log/munge/munged.log`

### 7.5.2 Installing, Configuring, And Running UGE

The workload manager, Univa Grid Engine (UGE), is a further development of the venerable SGE. NVIDIA Bright Cluster Manager 9.2 supports integration with UGE versions 8.6.0 and higher at the time of writing of this section (December 2019).

UGE should be picked up directly from the Univa website at [http://www.univa.com](http://www.univa.com).

The installation and integration of UGE into NVIDIA Bright Cluster Manager 9.2 can be carried out as in the following steps:

1. These UGE tar.gz file collections should be downloaded from the Univa website to a directory on the head node:
   - Binary files:
     - The 64-bit Linux bundle
       `ge-<uge_ver>-bin-1x-amd64.tar.gz`
     - or the 32-bit Linux bundle
       `ge-<uge_ver>-bin-1x-x86.tar.gz`
   - Common files:
     - `ge-<uge_ver>-common.tar.gz`

   Here `<uge_ver>` is the UGE version, for example: 8.6.0

   To avoid installation issues, a check should be done to ensure that during the download, the tar.gz files have not been renamed, or that their names not been changed to upper case. Both packages must be located in the same directory before installation.

   If a failover setup already exists, then the installation should be done on the active head node in the steps that follow.

2. The `cm-uge` package should be installed from the Bright Computing repository via the distribution’s package manager. That is, via `yum`, `zypper`, or `apt`, as appropriate.

   The package installs, amongst others, the following template files under `/cm/shared/apps/uge/var/cm/`:

   (a) An environment module template, `uge.module.template`

   (b) An installation configuration template, `inst_template.conf.template`

   (c) Some other template files for a default GE configuration, under the directory `templates`
The templates decide the configuration of UGE (section “Configuring UGE”, page 325).

The original files 2a and 2b, with the .template suffix, should never be modified by the administrator. The administrator can change the parameters affected by the installation configuration from their default values by copying these original files to the same directory without the .template suffix, and then editing the .conf file:

Example

```
[root@bright92 ~]# cd /cm/shared/apps/uge/var/cm
[root@bright92 cm]# cp inst_template.conf.template inst_template.conf
[root@bright92 cm]# vi inst_template.conf
```

The copied file, inst_template.conf, can be changed by the administrator to decide the location of UGE software and how UGE handles jobs, if needed. The changed file overrides the settings suggested by the .template file, when the cluster manager utility cm-wlm-setup runs in the next step of the installation procedure.

Some of the values in the key-value pairs in the file are enclosed by percentage signs, %. For example: CELL_NAME=\%CELL_NAME\%. Such flagged values should not be modified, since they are replaced by the cluster manager values by cm-wlm-setup during installation.

Values not enclosed by percentage signs are not replaced by cm-wlm-setup. Such unflagged values can, if needed, be tuned in the copied file by the administrator. These values are then kept by cm-wlm-setup during installation, and used when UGE is run.

3. cm-wlm-setup is run. The directory where the downloaded UGE files are located can be specified with the --archives-location option, if using the CLI mode with options.

Example

```
root@bright92:~# ls -al /root/uge/
total 67892
drwxr-xr-x 2 root root 82 Dec 18 16:03 .
```

```
root@bright92:~# cm-wlm-setup --wlm uge --setup --archives-location /root/uge
```

4. The nodes are rebooted. The UGE command qhost then displays an output similar to:

Example

```
[root@bright92 ~]# module load uge
[root@bright92 ~]# qhost
```

```
HOSTNAME     ARCH    NCPU  NSOC  NCOR  NTHR  NLOAD ...
-----------------------------------------------...
global       -       -       -       -       -       ...
node001      lx-amd64  8     1     1     8     0.01 ...
node002      lx-amd64  8     1     1     8     0.01 ...
node003      lx-amd64  8     1     1     8     0.01 ...
node004      lx-amd64  8     1     1     8     0.01 ...
```

The output in the preceding example has been truncated for this manual, for convenience.
Configuring UGE
After installation and initialization, UGE has reasonable defaults, with $SGE_ROOT set to /cm/shared/apps/uge/current.

By default, the UGE application is installed in /cm/shared/apps/uge/current, and job examples are kept in /cm/shared/examples/workload/sge/jobscripts/.

Running UGE
The UGE workload manager runs the following two daemons:

1. an sge_qmaster daemon running on the head node. This handles queue submissions and schedules them according to criteria set by the administrator.
2. an sge_execd execution daemon running on each compute node. This accepts, manages, and returns the results of the jobs on the compute nodes.

Messages from the qmaster daemon are logged in:

/cm/shared/apps/uge/current/<uge instance name>/spool/messages

On the associated compute nodes the execution log messages exists, alongside other job tracking files and directories, at:

/cm/local/apps/uge/var/spool/node<number>/messages

where node<number> is the node name, for example:

node001, node002 …

UGE Submission, Administrative And Execution Hosts
UGE is still similar to SGE in many ways, but is evolving.

In UGE terminology:

- A submission host, or submit host, is a node that is allowed to submit jobs. Submission can be from the user, or from another job run by the user.
- An execution host, or exec host, is a node on which the computational part of the job is executed.
- An administrative host, or admin host, is a node that can carry out administrative Grid Engine commands.

Standard Cluster Manager Use
In a standard cluster manager configuration, a submission, administrative or execution host in UGE is configured with the following ugeclient role properties:

1. Admin Host – adds the node to a list of administrative hosts in UGE,
2. Exec Host – adds the node to a list of execution hosts in UGE,
3. Submit Host – adds the node to a list of submission hosts in UGE.

The nodes that are assigned a ugeserver role are always automatically added to the administrative and submission host lists.

Non-standard Cluster Manager Use
For non-standard configurations, where the UGE service or login service is managed outside of the cluster manager control, advanced configuration directives allow the cluster to be aware of the situation.
1. If the ugeclient role cannot be added, for instance due to the UGE service being managed outside of the cluster manager, then the additional execution hosts can be specified in the advanced configuration directive, `AdditionalExecHosts`, in the `cmd.conf` file:

   **Example**
   ```
   AdvancedConfig = {"AdditionalExecHosts=node002,login01"}
   ```

2. If the ugeclient role cannot be added, for instance due to the login service being managed outside of the cluster manager, then the execution hosts can be specified in the advanced configuration directive, `AdditionalSubmitHosts`, in the `cmd.conf` file:

   **Example**
   ```
   AdvancedConfig = {"AdditionalSubmitHosts=node002,login01"}
   ```

### Parallel Environments

A parallel environment (PE) allows the configuration of applications that use shared or distributed memory. The main—but not sole—purpose of a PE is to provide a job environment within which MPI applications can run, using various MPI libraries. PEs can be provided by the cluster software or hardware vendor, as well as distributed with UGE itself. Default PEs are provided for various MPI libraries, such as Open MPI, MVAPICH, and so on. The default PEs are automatically added to UGE when `cm-wlm-setup` installs UGE.

The CMDaemon front ends, i.e. `cmsh` or Bright View, can be used to add, remove, or modify the existing PE. Each UGE cluster has a `Parallel Environments` submode. In the `Parallel Environments` submode, each PE is represented as a separate object with associated properties. For example, with `cmsh` the `wlm` mode is used to access the Open MPI properties within a UGE cluster (some text elided):

**Example**

```
[root@bright92 ~]# cmsh
[bright92]% wlm
[bright92->wlm]% list
Type Name (key) Server nodes Submit nodes Client nodes
------- ----------- ------------ -----------------------------
UGE uge bright92 node001..node003,bright92 node001..node003
[bright92->wlm[uge]]% use uge
[bright92->wlm[uge]]% parallelenvironments
[bright92->wlm[uge]]->parallelenvironments]% list
Name (key) Slots Allocation Rule
----------------- -------- ----------------
impi 999999 $round_robin
mpich 999999 $round_robin
mpich2 999999 $round_robin
openmpi 999999 $round_robin
openmpi_rankfile 999999 4
[bright92->wlm[uge]]->parallelenvironments]% use openmpi
[bright92->wlm[uge]]->parallelenvironments[openmpi]]% show
Parameter Value
-----------------------------------------------
Daemon Forks Slaves yes
Master Forks Slaves no
Revision
Type UGEParallelEnvironment
```
The values shown in the preceding example are defaults.

**Database Recovery**

By default, the cluster manager configures UGE to use flat file, or classic spooling as its format for the sge_qmaster spool. Typically this can handle thousands of jobs a day without running into performance issues, and it is the current (May 2015) recommendation for stability reasons.

For larger clusters it is possible to use Berkeley DB as the spooling format. This comes with a number of utilities, some of which can be useful for UGE spool maintenance and debugging, such as db_recover, db_verify, and others.

The Berkeley DB utilities that come with UGE are modified versions of the ones that are shipped with the parent distribution Berkeley DB packages. The UGE versions include extra sanity tests for the UGE spool database, and it is best to use them instead of the standard versions. The utilities can be found in on the head node under:

```
/cm/shared/apps/uge/current/utilbin/lx-amd64/
```

This directory is not added to the $PATH when loading the UGE environment with module load uge.

The full Berkeley DB documentation for these tools is part of the UGE distribution. A local HTML format copy of the documentation is available at:

```
/cm/shared/apps/uge/current/doc/berkeleydb/db_*.html
```

**GPU And MIC Management**

When managing host resources such as GPU or Xeon Phi (MIC) PCI-e cards, UGE 8.1.0 and higher allows the accelerators to be configured via a resource map, RSMAP. This is a complex (man (5) complex) value type defined on the host. RSMAP values restrict how much of a resource is used concurrently from a host, attaches identifiers to the resource used, and assigns the identifiers to the jobs when they get dispatched to that host. The cluster manager automatically configures UGE execution hosts with gpu and phi RSMAP resources.

The ugeclient role can configure GPUs by using the following cmsh parameters:

- **GPU devices**: a list of GPU names that are attached to a job by UGE.

---

1Integration of PCI-e MIC cards is discontinued from NVIDIA Bright Cluster Manager 8.2 onwards, but UGE support for the cards via RSMAP attributes continues. This means that the administrator typically would download the MIC MPSS packages, compile the driver, install the images manually, and after getting the MICs to work, would integrate the MIC with the appropriate RSMAP attributes.
• **Gpus**: the number of GPUs on a host.

The **GPU devices** parameter has a higher priority than **Gpus**, so that if names are set in **GPU devices**, then they are always used.

GPUs can also be bound to particular CPU sockets and CPU cores, using a **topology mask**.

The mask is defined with a set of characters. In principle, the mask is applied to the computing units available for a device. The computing units correspond to processing units, and are grouped as CPU sockets, cores, and hardware threads. These correspond to the letters S, C, T. In the mask. An uppercase character allows a device to use a computing unit, while a lowercase character forbids it, as indicated by the following table:

<table>
<thead>
<tr>
<th>Unit To Mask</th>
<th>Enable Unit with</th>
<th>Disable Unit with</th>
</tr>
</thead>
<tbody>
<tr>
<td>Socket</td>
<td>S</td>
<td>s</td>
</tr>
<tr>
<td>Core</td>
<td>C</td>
<td>c</td>
</tr>
<tr>
<td>Hardware Thread</td>
<td>T</td>
<td>t</td>
</tr>
</tbody>
</table>

In practice, s and t are currently (May 2015) ignored, so that only cores can be masked.

Some mask examples:

• **SccSCC**: This is a two-core two-socket system. The first socket cannot be used, while both cores on the second socket can be used.

• **SCcCCcSCC**: This is a four-core two-socket system. The first socket has its first, third, and fourth cores available, and the second socket has its third and fourth cores available.

Some configuration examples from cmsh and the corresponding RSMAP complex attribute values on the host:

• Naming 2 GPUs:

```
[root@bright92 ~]# cmsh
[bright92]# category roles default
[bright92->device[bright92]->roles] assign ugeclient; commit
[bright92->device[bright92]->roles[ugeclient]] show | grep -i gpu
GPU devices
GPUs 2
```

The RSMAP attribute value is then:

```
GPU=2(gpu0 gpu1)
```

• Allow GPUs to use only different sockets:

```
[bright92->device[bright92]->roles[ugeclient]] show | grep -i gpu
GPU devices
gpu0:SCcCCcSccc gpu1:ScccSCCcc
GPUs 0
```

The RSMAP attribute value is then:

```
GPU=2(gpu0:SCcCCcSccc gpu1:ScccSccCcc)
```

• Allow Xeon Phi resources to set up sockets in a spaced-out way:

Here, 4 MICs mic0 to mic3 have their cores enabled as follows: The first core of mic0, the second core of mic1, the third core of mic2 and the fourth core of mic3. The remaining cores of the MICs are disabled:
As is seen here, MICs are set up very much like GPUs, and also within the ugeclient role. The only differences are that the parameters that are set differ according to the accelerator used:

- while for GPUs topology masks are set for the GPU devices parameter, for MICs topology masks are set for the MIC devices parameter
- while for GPUs the number of GPUs is manually specified in the Gpus parameter, for MICs the number of MICs is counted automatically and there is no equivalent parameter.

The RSMAP attribute value in this example is then:

\[
\text{MIC}=4 (\text{mic0:SCccc mic1:ScCcc mic2:SccCc mic3:ScccC})
\]

Further detailed information about RSMAP and topology mask usage can be found in the Univa Grid Engine Administrator’s Guide.

### 7.5.3 Configuring And Running PBS

#### PBS Variants And Versions

The cluster manager is integrated with PBS Pro Commercial workload manager version 2021, and with the OpenPBS workload manager version 20.

- **PBS Pro.** This is the commercial variant. It requires a license or a license server in order to run jobs. This information can be provided during a run of the setup wizard, or it can be manually configured after setup.
  - The version 2021 packages are available as:
    * pbspro2021 for the server,
    * pbspro2021-client for the compute nodes.
  
  Pre-2020 major versions of this PBS variant were denoted by 2 digits (such as 18 or 19) to signify the year. Since version 20 the numbering is denoted by 4 digits (such as 2020 and 2021).

- **OpenPBS.** This is the open source variant with community support (http://openpbs.org). The community edition packages have a version 20 (with a 2-digit version, while PBS Pro nowadays uses 4-digit versions), and are available as:
  - openpbs20 for the server,
  - openpbs20-client for the compute nodes.

When no particular PBS variant is specified in the cluster manager documentation, then the text is valid for both variants. The cluster manager provides a similar level of integration for the commercial and the community packages. It is up to the cluster administrator to decide which variant is set up.

Both variants can be installed as a selection option during NVIDIA Bright Cluster Manager 9.2 installation, at the point when a workload manager must be selected (figure 3.10 of the Installation Manual). Alternatively they can be installed later on, when the cluster has already been set up.

The PBS packages that Bright Computing provides from the Bright Computing repositories should be used instead of other available versions, such as from the Linux distribution.

If the PBS packages themselves have not been picked up, they can be installed and removed with a package manager such as YUM.

#### Example

```
[root@bright92 ~]# yum install pbspro2021 pbspro2021-client
[root@bright92 ~]# yum install --installroot=/cm/images/default-image pbspro2021-client
```

If the cluster manager has already been set up without PBS, but with PBS packages installed via YUM, then the cm-wlm-setup tool (section 7.3) should be used to install and initialize PBS.
Installing PBS
After package installation via the package manager, as described in the preceding section, PBS can be installed and initialized to work with the cluster manager via `cm-wlm-setup`. With no options, an Ncurses session is started to guide the process. The alternative CLI process with options might take the following forms:

**Example**
```
[root@bright92 ~]# cm-wlm-setup --wlm pbspro --wlm-cluster-name ppro --license <license information>
```

or
```
[root@bright92 ~]# cm-wlm-setup --wlm openpbs --wlm-cluster-name opbs
```

The option `--wlm pbspro` installs the commercial version, while `--wlm openpbs` installs the community version.

The *license information* is either a path to a license file, or it is a Altair license server address list in the format:

```
<port1>@<host1>::<port2>@<host2>::...@<portN>@<hostN>
```

This license information can also be set manually for the `pbs_license_info` attribute. For example if there is just one license server, `pbspro-license-server`, serving on port 6200, it could be set with `qmgr` as follows:

**Example**
```
qmgr -c "set server pbs_license_info = 6200@pbspro-license-server"
```

The software components are installed and initialized by default under the Spool directory, which is defined by the PBS_HOME environment variable. The directory is named after the WLM cluster name, and follows a path of the form:

```
/cm/shared/apps/pbspro/var/spool/<WLM cluster name>
```

or
```
/cm/shared/apps/openpbs/var/spool/<WLM cluster name>
```
as appropriate.

The paths `/cm/shared/apps/pbspro` or `/cm/shared/apps/openpbs` are the Prefix settings of the WLM, and depend on whether the WLM is running PBS Pro, or OpenPBS.

Users must load an environment module associated with the cluster name to set `$PBS_HOME` and other environment variables, in order to use that cluster.

**Example**
```
[root@bright92 ~]# module load pbspro
```

**PBS Configuration**
PBS documentation is available at [https://www.altair.com/pbs-works-documentation/](https://www.altair.com/pbs-works-documentation/).

By default, PBS examples are available under the directory `/cm/shared/examples/workload/pbspro/jobscripts/`

Some PBS configuration under the cluster manager can be done using roles. The roles are the same for the variants of PBS, and are denoted as PBS Pro roles:

- In Bright View the roles setting allows the configuration of PBS Pro client and server roles.
  - For the PBS Pro server role, the role is enabled, for example on a head node `bright92`, via a clickpath of:
    ```
    Devices→Head Nodes→bright92→Settings→Roles→Add→PBS pro server role
    ```
Within the role window for the server role, its installation path and server spool path can be specified.

- For the PBS Pro client role, the role is enabled along a similar clickpath, just ending at PBS pro client role. The number of slots, GPUs and other properties can be specified, and queues can be selected.

In cmsh the client and server roles can be managed for the individual nodes in device mode, or managed for a node category in category mode, or they can be managed for a configuration overlay in configuration overlay mode.

For example, if there is a PBS Pro cluster instance called pbsfast, then the head node could be assigned a PBS Pro server role, with the following properties:

**Example**

```
[root@bright92 ~]# cmsh
[bright92] % device roles bright92
[bright92->device[bright92]->roles] % assign pbsproserver
[bright92->device*[bright92*]->roles*[pbsproserver*]] % show
```

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>Name</td>
<td>pbsproserver</td>
</tr>
<tr>
<td>Revision</td>
<td></td>
</tr>
<tr>
<td>Type</td>
<td>PbsProServerRole</td>
</tr>
<tr>
<td>Add services</td>
<td>yes</td>
</tr>
<tr>
<td>WLM cluster</td>
<td></td>
</tr>
<tr>
<td>Provisioning associations</td>
<td>&lt;0 internally used&gt;</td>
</tr>
<tr>
<td>External Server</td>
<td>no</td>
</tr>
<tr>
<td>Comm Settings</td>
<td>&lt;submode&gt;</td>
</tr>
</tbody>
</table>

```
[bright92->device*[bright92*]->roles*[pbsproserver*]] % set wlmcluster pbsfast; commit
```

Similarly, a category of nodes that can be used by the instance pbsfast, and called fastnodes, may have a PBS Pro client role assigned and set with the following properties:

**Example**

```
[bright92->category[fastnodes]->roles[pbsproclient]] % show
```

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>Name</td>
<td>pbsproclient</td>
</tr>
<tr>
<td>Revision</td>
<td></td>
</tr>
<tr>
<td>Type</td>
<td>PbsProClientRole</td>
</tr>
<tr>
<td>Add services</td>
<td>yes</td>
</tr>
<tr>
<td>WLM cluster</td>
<td>pbsfast</td>
</tr>
<tr>
<td>Slots</td>
<td>1</td>
</tr>
<tr>
<td>GPUs</td>
<td>0</td>
</tr>
<tr>
<td>All Queues</td>
<td>no</td>
</tr>
<tr>
<td>Queues</td>
<td></td>
</tr>
<tr>
<td>Properties</td>
<td></td>
</tr>
<tr>
<td>Provisioning associations</td>
<td>&lt;0 internally used&gt;</td>
</tr>
<tr>
<td>Mom Settings</td>
<td>&lt;submode&gt;</td>
</tr>
<tr>
<td>Comm Settings</td>
<td>&lt;submode&gt;</td>
</tr>
<tr>
<td>Node Customizations</td>
<td>&lt;0 in submode&gt;</td>
</tr>
</tbody>
</table>
Cloudbursting with cluster extension may need special handling for PBS. If this feature is needed, then the administrator should contact Bright Computing support via the website https://support.brightcomputing.com.

Further configuration of PBS is done using its `qmgr` command and is covered in the PBS documentation.

Running PBS

For the WLM cluster instances, PBS runs the following four daemons:

1. a `pbs_server` daemon running, typically on the head node. This handles submissions acceptance, and talks to the execution daemons on the compute nodes when sending and receiving jobs. It writes logs to the `var/spool/<WLM cluster instance>/server_logs/` directory, which is a directory that is under `/cm/shared/apps/pbspro` or `/cm/shared/apps/openpbs`. Queues for this service are configured with the `qmgr` command.

2. a `pbs_sched` scheduler daemon, also typically running on the head node. It writes logs to the `var/spool/<WLM cluster instance>/sched_logs/` directory under `/cm/shared/apps/pbspro` or `/cm/shared/apps/openpbs`.

3. a `pbs_mom` execution daemon running on each compute node. This accepts, manages, and returns the results of jobs on the compute nodes. By default, it writes logs
   - to the relative directory `var/spool/<WLM cluster instance>/mom_logs/`, which is under `/cm/shared/apps/pbspro` or `/cm/shared/apps/openpbs` and
   - to `/cm/local/apps/pbspro/var/spool/mom_logs/` on nodes with the client role.

4. a `pbs_comm` communication daemon usually running on the head node. This handles communication between PBS daemons, except for server-to-scheduler and server-to-server daemons communications. It writes logs
   - to the relative directory `var/spool/<WLM cluster instance>/comm_logs/`, which is under `/cm/shared/apps/pbspro` or `/cm/shared/apps/openpbs` and
   - to `/cm/local/apps/pbspro/var/spool/comm_logs/` on nodes with the client role.

Running PBS On Cluster Extension

When PBS is set up with `cm-wlm-setup` or during the installation of the head node, then the `pbsproclient` role is assigned by default via a configuration overlay to the default node category only.

    [root@bright92 ~]# cmsh -c "category; roles default; list"
    Name (key)  
    ----------------------------------------
    [overlay:openpbs-client] pbsproclient  
    ...

    In order to add cloud nodes to PBS, the administrator can assign the `pbsproclient` role manually. There are two types of PBS configuration in this case. The configurations can be applied to both the commercial and to the community editions.

1. `pbs_mom` daemons on cloud compute nodes communicate to the `pbs_server` directly.

   This scenario is suited to a non-VPN setup where cloud nodes have addresses on the same IP subnet for the cloud and for the on-premises parts of the cluster. Usually this kind of setup is used with Amazon DirectConnection or Azure ExpressRoute. In order to add new cloud nodes, the administrator just needs to assign the `pbsproclient` role to the cloud node category or the cloud nodes directly.
2. PBS mom daemons on cloud compute nodes communicate to the PBS server via a separate PBS comm server. Bright Computing recommends the use of the cloud-director for this purpose. This can be useful if cluster extension is configured with a VPN tunnel setup. In this case the PBS mom running on a cloud node communicates with the PBS server by using the VPN connection, and the communication traffic goes via an OpenVPN server. The OpenVPN connection adds overhead on the cloud-director where the OpenVPN daemon runs. If the traffic is routed via PBS comm running on cloud-director, then the OpenVPN server is not used. This is because PBS mom daemon on the cloud director resolves the cloud PBS mom addresses with cloud IP addresses, while PBS server resolves PBS comm on the cloud director by using the VPN tunnel IP.

In order to configure PBS to pass the communication traffic via PBS comm, the administrator should assign PBS pro client roles to not only the compute cloud nodes, but also to the cloud-director. On the cloud director, the administrator should enable PBS comm daemon to start, and PBS mom daemon to not start, automatically. These actions are done in the commsettings and momsettings submodes of PBS pro client role.

For the PBS pro client role assigned in the configuration overlay, the settings can be accessed in cmsh as in the following:

**Example**

```
[root@bright92 ]# cmsh
[bright92]# configurationoverlay
[bright92->configurationoverlay]# use openpbs-client
[bright92->configurationoverlay[openpbs-client]]# roles
[bright92->configurationoverlay[openpbs-client]-roles]# use pbsproclient
[bright92->configurationoverlay[openpbs-client]-roles[pbsproclient]]# show
Parameter Value
-------------------------------- ------------------------------------------------
Name pbsproclient
Revision
Type PbsProClientRole
Add services yes
WLM cluster openpbs
Slots 1
GPUs 0
All Queues no
Queues workq
Properties
Provisioning associations <0 internally used>
Mom Settings <submode>
Comm Settings <submode>
Node Customizations <0 in submode>
```

```
[bright92->configurationoverlay[openpbs-client]-roles[pbsproclient]]# commsettings
[bright92->configurationoverlay[openpbs-client]-roles[pbsproclient]-commsettings]# show
Parameter Value
-------------------------------- ------------------------------------------------
Comm Routers
Revision
Comm Threads 4
Start Comm no
```

```
[bright92->configurationoverlay[openpbs-client]-roles[pbsproclient]-commsettings]# ..
[bright92->configurationoverlay[openpbs-client]-roles[pbsproclient]]# momsettings
[bright92->configurationoverlay[openpbs-client]-roles[pbsproclient]-momsettings]# show
Parameter Value
-------------------------------- ------------------------------------------------
```
Further configuration that should be carried out is to set the `commrouters` parameter on the cloud director to `master`, and set the `leafrouters` parameter on the compute cloud nodes to the hostname of the cloud director.

For example (some text elided):

Example

```
[root@bright92 ~]# cmsh
[bright92] → category roles cloud-nodes
[bright92->category[cloud-nodes]->roles]% assign pbsproclient
[bright92->category[cloud-nodes]->roles[pbsproclient]]% set queues cloudq
[bright92->...]->roles[pbsproclient]->momsettings]% momsettings
[bright92->...]->roles[pbsproclient]->momsettings]% set leafrouters director
[bright92->...]->roles[pbsproclient]->momsettings]% commit
[bright92->...]->roles[pbsproclient]->momsettings]% device roles director
[bright92->...]->roles]% assign pbsproclient
[bright92->...]->roles[pbsproclient]]% momsettings
[bright92->...]->roles[pbsproclient]->momsettings]% set startmom no
[bright92->...]->roles[pbsproclient]->momsettings]% ..
[bright92->...]->roles[pbsproclient]->commsettings]% commsettings
[bright92->...]->roles[pbsproclient]->commsettings]% set startcomm yes
[bright92->...]->roles[pbsproclient]->commsettings]% set commrouters master
[bright92->...]->roles[pbsproclient]->commsettings]% commit
[bright92->...]->roles[pbsproclient]->commsettings]%
```

7.5.4 Installing, Configuring, And Running LSF

IBM prefers to make LSF available directly from their Passport Advantage Online website, which is why it is not available by direct selection in figure 3.10 of the *Installation Manual*.

Installing LSF

The workload manager LSF version 10.1 is installed and integrated into NVIDIA Bright Cluster Manager 9.2 with the following steps:

1. The following LSF files should be downloaded from the IBM web site into a directory on the head node:
   - Installation package: `lsf<lsf_ver>_lsfinstall_linux_<cpu_arch>.tar.Z`
   - Distribution package: `lsf<lsf_ver>_linux_<kern_ver>-glibc<glibc_ver>-<cpu_arch>.tar.Z`
   - Documentation package (optional): `lsf<lsf_ver>_documentation.tar.Z`

Here:

- `<lsf_ver>` is the LSF version, for example: `10.1`
- `<kern_ver>` is the Linux kernel version, for example: `2.6`
7.5 Configuring And Running Individual Workload Managers

- `<glibc_ver>` is the glibc library version, for example: 2.3
- `<cpu_arch>` is the CPU architecture, for example: x86_64

A check should be done to ensure that the `tar.Z` files have not been renamed or had their names changed to lower case during the download, in order to avoid installation issues. All the files must be in the same directory before installation.

In case of an existing failover setup, the installation is done on the active head node.

A license file for LSF may also be needed for the installation, but it does not have to be in the same directory as the other `tar.Z` files.

2. The `cm-lsf` package must be installed on the head node. The `cm-lsf-client` package must be installed on both the head node and within the software images. By default they should have been already installed. If not then they can be installed manually from the Bright Computing repository.

For RHEL-based distributions the procedure looks like:

```
[root@bright92 ~]# yum install cm-lsf cm-lsf-client
[root@bright92 ~]# chroot <IMAGE> yum install cm-lsf-client
```

For SLES distributions the procedure looks like:

```
[root@bright92 ~]# zypper install cm-lsf cm-lsf-client
[root@bright92 ~]# chroot <IMAGE> zypper install cm-lsf-client
```

For Ubuntu distributions the procedure looks like:

```
[root@bright92 ~]# apt-get install cm-lsf cm-lsf-client
[root@bright92 ~]# cm-chroot-sw-img <IMAGE>
[root@bright92 ~]# apt-get install cm-lsf-client
...
[root@bright92 ~]# exit
```

The `cm-lsf` and `cm-lsf-client` packages contain a template for an environment module file, an installation configuration file, and systemd unit files. The installation configuration file may be tuned by the administrator if required. It is passed to the `lsfinstall` script distributed with LSF, which is executed by `cm-wlm-setup` during setup. To change the default values in the installation configuration file, the administrator should change the template file:

```
[root@bright92 ~]# cd /cm/shared/apps/lsf/var/cm/
[root@bright92 cm]# vi install.config.template
...
```

Values enclosed by a percentage sign, `%`, are replaced by `cm-wlm-setup` during installation. If such values are replaced by custom values, then `cm-wlm-setup` does not change them, and the custom values are used during installation.

If `install.config` is changed instead of the template file, then `cm-wlm-setup` replaces it with the configuration file generated from the template. So, changing `install.config` directly should almost certainly not ever be done.

3. `cm-wlm-setup` is run. The directory where the LSF files were downloaded is specified on one of the setup screens, or with the `--archives-location` option.

**Example**
The same can be achieved by executing `cm-wlm-setup` without any command line arguments. The required information can be specified within the Ncurses configuration screens in this case. Also the same can be achieved with the WLM Wizard in Bright View.

4. The nodes are then rebooted, and the LSF command `bhosts` then displays an output similar to:

```
Example

[root@bright92 ~]# module load lsf
[root@bright92 ~]# bhosts
HOST_NAME STATUS JL/U MAX NJOBS RUN SSUSP ...
bright92 ok 2 0 0 0 ...
head2 ok 0 0 0 0 ...
node001 ok 1 0 0 0 ...
node002 ok 1 0 0 0 ...
node003 avail 1 0 0 0 ...
node004 closed 1 0 0 0 ...
node005 ok 1 0 0 0 ...
```

The output in the preceding example has been truncated for this manual, for convenience.

The installation status can be checked with the service `lsfd` (some output elided):

```
[root@bright92 ~]# systemctl status lsfd
lsfd.service - IBM Spectrum LSF
    Loaded: loaded (/usr/lib/systemd/system/lsfd.service; enabled; vendor preset: disabled)
    Active: active (running) since Mon 2020-03-30 13:59:29 CEST; 27min ago
    Tasks: 13
    Memory: 107.6M
    CGroup: /system.slice/lsfd.service
        |- 2411 /cm/shared/apps/lsf/lsf...3-x86_64/10.1/linux2.6-glibc2.3-x86_64/etc/lim
        |- 3470 /cm/shared/apps/lsf/lsf...3-x86_64/10.1/linux2.6-glibc2.3-x86_64/etc/pim
        |- 3472 /cm/shared/apps/lsf/lsf...3-x86_64/10.1/linux2.6-glibc2.3-x86_64/etc/record
        |- 3478 /cm/shared/apps/lsf/lsf...3-x86_64/10.1/linux2.6-glibc2.3-x86_64/etc/sbatchd
        |- 3483 /cm/shared/apps/lsf/lsf...3-x86_64/10.1/linux2.6-glibc2.3-x86_64/etc/mbatchd -d
        |    /cm/shared/apps/l...
        |- 3499 /cm/shared/apps/lsf/lsf...3-x86_64/10.1/linux2.6-glibc2.3-x86_64/etc/mbschd
        |- 3527 /cm/shared/apps/lsf/lsf...3-x86_64/10.1/linux2.6-glibc2.3-x86_64/etc/melim
        |   -11580 /cm/shared/apps/lsf/lsf...3-x86_64/10.1/linux2.6-glibc2.3-x86_64/etc/mbatchd -d
        |      /cm/shared/apps/l...
Mar 30 13:59:21 bright92 systemd[1]: Starting IBM Spectrum LSF...
Mar 30 13:59:22 bright92 lsf_daemons[2274]: Starting the LSF subsystem
Mar 30 13:59:29 bright92 systemd[1]: Started IBM Spectrum LSF.
```

while default queues can be seen by running:

```
Example

[root@bright92 ~]# module load lsf
[root@bright92 ~]# bqueues
QUEUE_NAME PRIU STATUS MAX JL/U JL/P ...
owners 43 Open:Active - - - ...
priority 43 Open:Active - - - ...
```
7.5 Configuring And Running Individual Workload Managers

The output in the preceding example has been truncated for this manual, for convenience.

If more than one instance of an LSF cluster is set up, then the full modulefile name should be specified. For example:

```
[root@bright92 ~]# module load lsf/lsf1/10.1
```

Configuring LSF

**LSF server configuration:** After LSF is set up with `cm-wlm-setup`, the following CMDaemon settings can be modified for the LSF server role:

- **External Server**: a value of `yes` means that LSF server daemons are running on an external server that is not managed by the cluster manager.

The following settings are available in LSF cluster settings, and are applied to all the LSF roles of the LSF cluster:

- **Prefix**: this sets the path to the root location of the LSF installation. The default value is `/cm/shared/apps/lsf/current`.

- **Var**: this sets the path to the `var` directory of LSF. The default value is `/cm/shared/apps/lsf/var`.

- **Cgroups**: this is a submode that contains LSF-related cgroups settings.

The cgroups settings that affect LSF behavior are available via the LSF instance. For example, for an instance `lsf`, the settings can be accessed within `cmsh` via `cmsh->wlm[lsf]->cgroups`. The settings available are:

- **Automount**: if `yes`, then the workload manager tries to mount a subsystem if it is not mounted yet. The default value is `No`.

- **Job Cgroup Template**: this is the template relative job cgroup path. The token `$CLUSTER` specified in this template path is replaced with the actual LSF cluster name, and `$JOBID` is replaced by the job ID. The path is used by the cluster manager monitoring system in order collect job metrics from the cgroups. By default, the path is set to "lsf/$CLUSTER/job.$JOBID.*".

- **Process Tracking**: if `yes`, then processes are tracked based on job control functions such as: termination, suspension, resume, and other signaling. These are used on Linux systems that support the freezer subsystem under cgroups. The parameter sets `LSF_PROCESS_TRACKING` in `lsf.conf`.

- **Linux Cgroup Accounting**: if `yes`, then LSF tracks processes based on CPU and memory accounting. This is for Linux systems that support the memory and cpuacct subsystems under cgroups. Once enabled, this parameter takes effect for new jobs. The parameter sets `LSF_LINUX_CGROUP_ACCT` in `lsf.conf`.

If this parameter and Process Tracking are both enabled, then they take precedence over the parameters LSF_PIM_LINUX_ENHANCE and EGO_PIM_SWAP_REPORT in `lsf.conf`.

- **Mount Point**: specifies a path where cgroups is mounted. It only makes sense to set this when the location is not standard for the operating system.
- **Resource Enforce**: If `yes`, then resource enforcement is carried out through the Linux memory and cgroup subsystems. This is for Linux systems with cgroup support. The parameter sets `LSB_RESOURCE_ENFORCE` in `lsf.conf`.

The server role settings can be modified as follows:

- **Within Bright View**: For example for a head node `bright92`, via a clickpath of
  Devices→Head Nodes→bright92→Settings→Roles→LSF server role

- **Within cmsh**: For a particular category in the category mode, or a particular device in the device mode, the `roles` submode is chosen. Within the `roles` submode, the `lsfserver` object can be assigned or used. The following example shows how to set the LSF prefix parameter for the default category.

  **Example**

  ```bash
  [root@bright92 ~]# cmsh
  [bright92]# configurationoverlay use lsf-server; roles
  [bright92->configurationoverlay[lsf-server]->roles]% use lsfserver
  [bright92->...[lsfserver]]% set externalserver yes
  [bright92->...[lsfserver]]% commit
  ```

The global LSF cluster settings can be modified as follows:

- **Within Bright View**:  For example, for a head node `bright92`, via a clickpath of:
  HPC→Wlm Clusters→CLUSTER_NAME→Settings

- **Within cmsh**:  A particular LSF cluster instance can be chosen within `wlm` mode, to set its global parameters. The following example shows how to set the LSF prefix parameter for the `lsf1` cluster.

  **Example**

  ```bash
  [root@bright92 ~]# cmsh
  [bright92]# wlm use lsf1
  [bright92->wlm[lsf1]]% set prefix /cm/shared/apps/lsf2
  [bright92->wlm[lsf1]]% commit
  ```

**LSF client configuration**: After installation, the following CMDaemon settings can be specified for the LSF client role:

- **Queues**  A restricted list of queues named “qname1”, “qname2” and so on can be set using a command syntax like this instead:

  ```bash
  set queues <qname1> [<qname2> ...]
  ```

  Alternatively, these, and other queues can be added or deleted using Bright View (section 7.6.2) or cmsh (section 7.7.2).
7.5 Configuring And Running Individual Workload Managers

• **Options**  The options that can be set for a regular node `node001`, via figure 7.13, reached via a clickpath of `Devices → Nodes[node001] → Edit → Settings → Roles → Add → LSF client role` are:

  - **Slots**: The number of CPUs per node. By default LSF tries to determine the value automatically. If the number of slots is set to 0, then the node becomes an LSF submit host, so that no jobs can run on the host, but users can submit their jobs from this host using the LSF utilities.
  - **GPUs**: The number of GPUs per node
  - **Queues**: All queues.
  - **Advanced button**: This brings up a dialogue that allows GPU devices to be added.
  - **Node Customizations**: LSF node custom properties (section 7.11).

From `cmsh` these properties are accessible from within the appropriate node or category `roles` submode (section 7.4.2).

**Submission hosts**: By default, all compute nodes to which the `lsfclient` role is assigned, also become submission hosts. This means that LSF users can submit their jobs from those compute nodes. If the administrator wants to allow users to submit their jobs from a non-compute node—for example from a login node—then the `lsfsubmit` role should be assigned, for example:

**Example**

```
[root@bright92 ~]# cmsh
[...->configurationoverlay[lsf1-submit]]% roles
[...->roles]% assign lsfsubmit
[...->roles*[lsfsubmit*]]% append lsfclusters lsf1
[...->roles*[lsfsubmit*]]% commit
[...->roles*[lsfsubmit*]]%
```

Configuring a node according to the preceding steps then allows users to submit their jobs from that node, without the submitted jobs getting scheduled to run on that node.

If more than one LSF cluster is set up, and they share the same submit host, then a single `lsfsubmit` role can be used. In this case, all the LSF cluster names should be appended to the `lsfcluster` parameter of the role.
Further configuration: For further configuration the *Administering Platform LSF* manual provided with the LSF software should be consulted.

Running LSF
Role assignment and role removal enables and disables LSF from Bright View (sections 7.4.1) or `cmsh` (section 7.4.2).

An active LSF master service (typically, but not necessarily on a head node) has the following LSF-related processes running on it:

<table>
<thead>
<tr>
<th>Process/Service</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>res</td>
<td>Remote Execution Server*</td>
</tr>
<tr>
<td>sbatchd</td>
<td>client batch job execution daemon*</td>
</tr>
<tr>
<td>mbatchd</td>
<td>master batch job execution daemon</td>
</tr>
<tr>
<td>eauth</td>
<td>External Authentication method</td>
</tr>
<tr>
<td>lim</td>
<td>Load Information Manager*</td>
</tr>
<tr>
<td>pim</td>
<td>Process Information Manager*</td>
</tr>
<tr>
<td>pem</td>
<td>Process Execution Manager*</td>
</tr>
<tr>
<td>vemkd</td>
<td>Platform LSF Kernel Daemon</td>
</tr>
<tr>
<td>egosc</td>
<td>Enterprise Grid Orchestrator service controller</td>
</tr>
<tr>
<td>mbschd</td>
<td>master batch scheduler daemon</td>
</tr>
</tbody>
</table>

*These services/processes run on compute nodes.

Non-active LSF-masters running as compute nodes run the processes marked with an asterisk only. LSF daemon logs are kept under `/cm/local/apps/lsf/var/log/`, on each of the nodes where LSF services run.

7.6 Using Bright View With Workload Management
Viewing the workload manager services from Bright View is described in section 7.4.3. The HPC (High Performance Computing) icon, which looks like a speedometer, is the Bright View resource that allows the following items to be accessed:

- WLM clusters: for WLM cluster settings for a WLM cluster instance to be viewed and managed
- jobs: for jobs to be viewed and managed
- queues: for queues to be viewed and managed
- job queue stats: for job queue statistics to be viewed

These items are described next.

7.6.1 Jobs Display And Handling In Bright View
The clickpath `HPC→Jobs` opens a window that displays a list of recent job IDs, along with the scheduler, user, queue, and status of the job (figure 7.14).
7.6 Using Bright View With Workload Management

If there are jobs running, then buttons in the OPTIONS column allow the jobs to be examined and managed.

- the Show button opens up a window with further details of the selected job.
- A selection button, , allows the selected job to be managed, via a Manage hint, using menu options as follows:
  - The Hold option stops selected queued jobs from being considered for running by putting them in a Hold state.
  - The Release option releases selected queued jobs in the Hold state so that they are considered for running again.
  - The Suspend option suspends selected running jobs.
  - The Resume option allows selected suspended jobs to run again.
  - The Cancel option removes selected jobs from the queue.

### 7.6.2 Queues Display And Handling In Bright View

The clickpath HPC→Wlm Clusters→Edit→Job Queues displays a list of queues available (figure 7.15).

![Figure 7.15: Workload Manager Queues](image)

Queues can be added or deleted. An Edit button allows existing queue parameters to be set, for example as in figure 7.16.
Using cmsh With Workload Management

The wlm mode in cmsh gives the administrator access to the WLM instances running on the cluster. The cmsh tree (Appendix M) has the following submodes structure under the wlm mode:

```
`-- wlm
    |--- cgroups
    |--- chargeback
    |--- jobqueue
    |--- jobs
    |--- parallelenvironments
    `-- placeholders
```

- cgroups: provides ways to configure WLMs via the cgroup mechanism of the Linux kernel (section 7.10)
- chargeback: provides ways to measure the costs of requested IT resources for WLM jobs (Chapter 16).
- jobqueue: allows WLM job queues to be managed (section 7.7.2)
- jobs: allows WLM jobs to be viewed and managed (section 7.7.1)
- parallelenvironments: allows MPI job environment parameters to be configured (page 326) for WLMs
7.7 Using cmsh With Workload Management

- **placeholders**: allows nodes to be configured in advance, for planning resource use in WLMs (section 8.4.8)

For basic WLM management via cmsh, the administrator is expected to use `wlm` mode with the `jobs` and `jobqueue` submodes.

If there is just one WLM instance running on the cluster, then an administrator that accesses `wlm` mode drops straight into the object representing that WLM instance, and can then access the submodes.

If there is more than one WLM running, then the administrator must select a WLM instance from the top level `wlm` mode, with the `use` command, in order to access the submodes.

While at the top level `wlm` mode, the instances can be listed. The list shows the nodes used by the instance, and the node roles assigned to the nodes.

Suppose the cluster manager is being used to manage two WLM instances:

- one Slurm WLM, named `slr`
- one UGE WLM, named `uc`

with their nodes having been allocated roles as follows:

**Example**

```
[bright92->wlm]% list
Type Name (key) Server nodes Submit nodes Client nodes
------ ------------------------ ------------ ------------------------- ----------------
Slurm slr node001 bright92,node001..node003 node002,node003
UGE uc node004 bright92,node004..node006 node005,node006
```

Then one way to list all jobs running per WLM instance, queue, and user, is with the `filter` command:

```
[bright92->wlm]% filter --running
WLM Job ID Job name User Queue Submit time Start time End time Nodes Exit code
---- ------ --------- ----- ------ ----------- ------------ --------- ----------------- ----------
slr 103 sjob maud defq 15:43:02 15:43:03 N/A node002,node003 0
slr 104 sjob maud defq 15:43:02 15:43:25 N/A node002,node003 0
slr 105 sjob maud defq 15:43:02 15:43:33 N/A node002,node003 0
slr 106 sjob maud defq 15:43:02 15:43:36 N/A node002,node003 0
slr 171 sjob maud defq 14:20:09 14:20:23 N/A node002,node003 0
uc 200 ugjob fred all.q 14:22:18 14:23:20 N/A node005,node006 0
```

The `filter` command can also be run within the `jobs` mode, for a particular WLM instance, and is described in more detail on page 345.

An alternate way to list all jobs from `wlm` mode is with a `foreach` command to descend into `jobs` submode to list the jobs for each WLM instance:

**Example**

```
[bright92->wlm]% foreach * (jobs ; list)
Type Job ID User Queue Running time Status Nodes
------ ------- ----- ------ ------------ ---------- ----------------
Slurm 55 maud defq 18m 10s COMPLETED node002,node003
Slurm 56 maud defq 18m 5s COMPLETED node002,node003
Slurm 57 maud defq 2m 11s RUNNING node002,node003
Slurm 58 maud defq 0s PENDING
```

```
Type Job ID User Queue Running time Status Nodes
------ ------- ----- ------ ------------ ---------- ----------------
UGE 96 maud all.q 2m 30s r node005,node006
UGE 97 maud 0s qw
UGE 98 maud 0s qw
```
The jobs submode is now discussed further.

### 7.7.1 The jobs Submode In cmsh

Within the jobs submode of a WLM instance, the administrator can list jobs that are running or queued up for that particular instance. For example, the running and queued jobs listed for a PBS instance may be displayed as:

**Example**

```plaintext
[bright92->wlm[openpbs]->jobs]% list
Type Job ID User Queue Running time Status Nodes
--------- ------------- ----------- ------------ ------------ -------- ---------------
PBSPro 3117.bright92 pbuser1 hydroq 1s R node002
PBSPro 3118.bright92 pbuser2 workq 1s R node001
PBSPro 3119.bright92 pbuser1 hydroq 0s Q
PBSPro 3120.bright92 pbuser3 hydroq 0s Q
PBSPro 3121.bright92 pbuser1 hydroq 0s Q
PBSPro 3122.bright92 pbuser1 hydroq 0s Q
```

In the foreach jobs listings for the Slurm and UGE WLM instances shown earlier, the jobs are in a running, queued up, or completed state. They are also all being run by a user maud, who happens to be making use of both workload managers.

For Slurm, completed jobs are shown for a short time with the status COMPLETED. For UGE, completed jobs are shown for an even shorter time with a blank for their status.

**Commands That Change The Status Of A Queued Or Running Job**

Within a jobs submode, the following commands, used with a job ID, can change the status of a particular queued or running job:

- **hold**: puts a queued job into a hold state. This prevents the job from being considered for running.
- **release**: releases a job from a hold state, putting it back in the queue, so that it can be considered for running.
- **suspend**: pauses a running job
- **resume**: resumes a suspended job
- **remove**: removes a job

For example, continuing the case earlier on of the cmsh session with the foreach listing (page 343):

The administrator can **suspend** the running job with job ID 57 on the Slurm instance as follows:

**Example**

```plaintext
[bright92->wlm]% jobs slr; suspend 57
[bright92->wlm[slr]->jobs]% list | head -2; list | grep 57
Type Job ID User Queue Running time Status Nodes
------- ------- ------ ------------ ---------- ----------------
Slurm 57 maud defq 2m 51s SUSPENDED node002,node003
```

and can **hold** the queued waiting job with job ID 98 on the UGE instance as follows:

**Example**
The administrator can then resume the suspended job ID 57, and can release the held job ID 98 as follows:

Example

Example

Commands To Inspect Jobs:
Within the jobs submode, the following commands can be used to inspect a job:

- `show <job ID>`: properties are shown for a running or pending job.
- `info <job ID>`: information is shown about a historic job.
- `statistics`: statistics are shown for jobs. More detail on the statistics command is given on page 346.
- `list`: lists pending and running jobs. Recently completed jobs are also displayed.
- `dumpmonitoringdata [OPTIONS] [<start-time> <end-time>] <metric> <job id>`: Displays the monitoring data for a specific metric or healthcheck for a running or completed job. Example: `dumpmonitoringdata CtxtSwitches 170`
  More detail is given on the dumpmonitoringdata command in section 13.6.4.
- `latestmonitoringdata [OPTIONS]<job ID>`: Displays the last measured job monitoring data for the job.
  - `latesthealthdata <job ID>`: The subset of health checks within the last measured monitoring data.
  - `latestmetricdata <job ID>`: The subset of metrics within the last measured monitoring data.
  More detail is given on the latest*data commands in section 13.6.3.
- `measurables <job ID>`: The measurables used for job monitoring data.
  - `healthchecks <job ID>`: The subset of health checks within the measurables used for job monitoring data.
  - `metrics <job ID>`: The subset of metrics within the measurables used for job monitoring data.
  - `enumetrics <job ID>`: The subset of enumetrics within the measurables used for job monitoring data.

The filter Command Within jobs Submode
If run at the top-level wlm mode, then the filter command lists jobs for all WLM instances.

It can also be run at the submode-level jobs mode, for a particular WLM instance, in which case it lists only the jobs for that instance (section 14.5).

The filter command without any options is a bit like an extension of the list command in that it lists currently pending and running jobs, although in a different format. However, in addition, it also lists past tasks, with their start and end times.

An example to illustrate the output format for a Slurm instance, somewhat simplified for clarity, is:
### Example

```bash
dir92->wlm[myslurminstance]]% filter
```

<table>
<thead>
<tr>
<th>Job ID</th>
<th>Job name</th>
<th>User</th>
<th>Queue</th>
<th>Submit time</th>
<th>Start time</th>
<th>End time</th>
<th>Nodes</th>
<th>Exit code</th>
</tr>
</thead>
<tbody>
<tr>
<td>2</td>
<td>hello</td>
<td>fred</td>
<td>defq</td>
<td>15:00:51</td>
<td>15:00:51</td>
<td>15:00:52</td>
<td>node001</td>
<td>0</td>
</tr>
<tr>
<td>3</td>
<td>mpirun</td>
<td>fred</td>
<td>defq</td>
<td>15:03:17</td>
<td>15:03:17</td>
<td>15:03:18</td>
<td>node001</td>
<td>0</td>
</tr>
<tr>
<td>...</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>25</td>
<td>slurmhello.sh</td>
<td>fred</td>
<td>defq</td>
<td>16:17:30</td>
<td>16:17:31</td>
<td>16:17:33</td>
<td>node001..node004</td>
<td>0</td>
</tr>
<tr>
<td>26</td>
<td>slurmhello.sh</td>
<td>fred</td>
<td>defq</td>
<td>16:18:39</td>
<td>16:19:02</td>
<td>16:19:25</td>
<td>node001..node004</td>
<td>0</td>
</tr>
<tr>
<td>27</td>
<td>slurmhello.sh</td>
<td>fred</td>
<td>defq</td>
<td>16:18:41</td>
<td>16:19:03</td>
<td>16:19:25</td>
<td>node001..node004</td>
<td>0</td>
</tr>
<tr>
<td>28</td>
<td>slurmhello.sh</td>
<td>fred</td>
<td>defq</td>
<td>16:18:41</td>
<td>N/A</td>
<td>N/A</td>
<td>node001..node004</td>
<td>0</td>
</tr>
<tr>
<td>29</td>
<td>slurmhello.sh</td>
<td>fred</td>
<td>defq</td>
<td>16:18:42</td>
<td>N/A</td>
<td>N/A</td>
<td>node001..node004</td>
<td>0</td>
</tr>
</tbody>
</table>

In the preceding example, the start and end times for jobs 28 and 29 are not yet available because the jobs are still pending.

Because all jobs for that WLM instance—historic and present—are displayed by the `filter` command, it means that finding a particular job in the output displayed can be hard. The displayed output can therefore be filtered further.

The following options to the `filter` command are therefore available in order to find particular groupings of jobs:

- `-w|--wlm`: by WLM instance
- `-u|--user`: by user
- `-g|--group`: by group
- `--ended`: by jobs that ended successfully, with a zero exit status
- `--running`: by jobs that are running
- `--pending`: by jobs that are pending
- `--failed`: by jobs that finished unsuccessfully, with a non-zero exit status

It may take about a minute for CMDaemon to become aware of job data. This means that, for example, a job submitted 10s ago may not show up in the output.

Further options to `filter` command can be seen by running the command `help filter` within the `wlm` or `jobs` modes.

### The statistics Command

The `statistics` command without any options displays an overview of states for all past and present jobs:

```bash
dir92->wlm[uc]]% statistics
```

<table>
<thead>
<tr>
<th>Pending</th>
<th>Running</th>
<th>Finished</th>
<th>Error</th>
<th>Nodes</th>
</tr>
</thead>
<tbody>
<tr>
<td>4</td>
<td>0</td>
<td>154</td>
<td>5</td>
<td>309</td>
</tr>
</tbody>
</table>

The options available include the following:

- `-w|--wlm <WLM instance>`: by WLM instance
- `-u|--user`: by user
- `-g|--group`: by group
7.7 Using cmsh With Workload Management

- `-a|--account`: by account
- `-p|--parentid`: by job parent ID
- `--hour`: by hour
- `--day`: by day
- `--week`: by week
- `--interval`: by interval

The jobs statistics can be split across users. An example of the output format in this case, somewhat simplified for clarity, is:

Example

```bash
[bright92->wlm[slurm]->jobs]% statistics --user
User       Pending  Running  Finished  Error  Nodes
----------- ---------- ---------- ---------- ----------
alice      0         0         2         0        1
bob        0         0         5         0        4
charline   0         1         2         0        2
dennis     0         0         6         0        4
eve        0         0         4         0        1
frank      0         0         1         0        1
```

The job statistics can be displayed over various time periods, if there are jobs within the period. An example of the output format for an interval of 60s is:

Example

```bash
[bright92->wlm[slurm]->jobs]% statistics --interval 60
Date       Pending  Running  Finished  Error  Nodes
---------- ---------- ---------- ---------- ----------
2020/05/13 15:00:00 0         0         1         0        1
2020/05/13 15:03:00 0         0         7         0        7
2020/05/13 15:04:00 0         0         1         1        2
2020/05/13 15:06:00 0         0         3         0        7
2020/05/13 15:07:00 0         0         1         0        0
2020/05/13 15:08:00 0         0         6         0        8
2020/05/13 16:17:00 0         0         1         0        4
2020/05/13 16:19:00 0         0         9         0       36
2020/05/13 16:20:00 0         0         6         0       24
2020/05/13 16:21:00 0         0         3         0       12
```

Job Directives

Job directives configure some of the ways in which CMDaemon manages job information processing. The administrator can configure the following directives:

- `JobInformationKeepCount`: The maximal number of jobs that are kept in the cache, default 8192, maximal value 100 million (page 871).
- `JobInformationKeepDuration`: How long to keep jobs in the cache, default 28 days (page 870).
- `JobInformationMinimalJobDuration`: Minimal duration for jobs to place them in the cache, default 0s (page 871).
- `JobInformationFlushInterval`: Over what time period to flush the cache to storage (page 871).
- `JobInformationDisabled`: Disables job information processing (page 870).
7.7.2 Job Queue Display And Handling In cmsh: jobqueue Mode

The jobqueue submode under the top level wlm mode can be used to manage queues for a WLM instance. Queue properties can be viewed and set.

From the top level wlm mode, a list of queues can be seen for each WLM instance by descending into the WLM instance with a foreach:

**Example**

```
[bright92->wlm]% foreach * (get name; jobqueue; list)
slr
Name (key)    Nodes
------------ ------------------------
defq           node002,node003
uc
Name (key)    Nodes
------------ ------------------------
all.q          node005,node006
[bright92->wlm]%
```

The usual object manipulation commands (show, get, set and others) of section 2.5.3 work in the jobqueue submode.

**Job Queue Parameters For UGE**

For example, for a UGE instance, the show command might display the following properties:

**Example**

```
[bright92->wlm[uc]->jobqueue]% show all.q
Parameter                  Value
---------------------------------------------------------------
Epilog                    NONE
Maximal runtime           INFINITY
Minimal runtime           INFINITY
Name                      all.q
Nodes                     node005,node006
Options                   Parallel environments: openmpi,mpich2
                         Prolog          NONE
Revision                  Revision
Slots                     0
Temp directory            /tmp
Type                      UGE
WlmCluster                uc
Resume method             NONE
Starter method            NONE
Suspend method            NONE
Terminate method          NONE
```

**Job Queue Parameters For Slurm**

Similarly, for a Slurm instance, the show command might display the following properties:

```
[bright92->wlm[slr]->jobqueue]% show defq
Parameter                  Value
---------------------------------------------------------------
Allow QOS                   ALL
Allow accounts              ALL
```
Allow groups ALL
Alternate
Default memory per CPU UNLIMITED
Default memory per Node UNLIMITED
Default time NONE
Deny Accounts
Deny Groups
Deny QOS
Disable root no
Grace time 0
Hidden no
LLN no
Max CPUs per node UNLIMITED
Max memory per CPU UNLIMITED
Max memory per Node UNLIMITED
Max nodes UNLIMITED
Max time UNLIMITED
Min nodes 1
Name defq
Nodes node002,node003
Options
OverSubscribe NO
Preemption mode OFF
Priority Job Factor 1
Priority Tier 1
Require reservation NO
Revision
Root only no
Select Type Parameters
Status
TRES Billing Weights
Total cpus 0
Total nodes 0
Type Slurm
WlmCluster slr
default yes

Job Queue Parameters For PBS
Likewise, for a PBS instance, the show command might display the following properties:

```
[bright92->wlm[pb]->jobqueue]% show workq
```

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>ACL host enable</td>
<td>no</td>
</tr>
<tr>
<td>Default Queue</td>
<td>yes</td>
</tr>
<tr>
<td>Default runtime</td>
<td></td>
</tr>
<tr>
<td>Enabled</td>
<td>yes</td>
</tr>
<tr>
<td>From Route Only</td>
<td>no</td>
</tr>
<tr>
<td>Maximal Queued</td>
<td>0</td>
</tr>
<tr>
<td>Maximal runtime</td>
<td>240:00:00</td>
</tr>
<tr>
<td>Minimal runtime</td>
<td>00:00:00</td>
</tr>
<tr>
<td>Name</td>
<td>workq</td>
</tr>
<tr>
<td>Nodes</td>
<td>node008,node009</td>
</tr>
<tr>
<td>Options</td>
<td></td>
</tr>
<tr>
<td>Priority</td>
<td>0</td>
</tr>
<tr>
<td>Queue Type</td>
<td>EXECUTION</td>
</tr>
</tbody>
</table>
### 7.7.3 Nodes Drainage Status And Handling In cmsh

Running the `device` mode command `drainstatus` displays if a specified node is in a Drained state or not. In a Drained state jobs are not allowed to start running on that node.

Running the `device` mode command `drain` puts a specified node in a Drained state:

**Example**

```
[root@bright92 ~]# cmsh
[bright92]# device
[bright92->device]# drainstatus
Node Queue Status
------------------------ ------------------------ ----------------
node001 PbsPro/pb:workq
node002 UGE/uc/all.q
[bright92->device]# drain node001
Node Queue Status
------------------------ ------------------------ ----------------
node001 PbsPro/pb:workq Drained
```

The `undrain` command unsets the Drained state so that jobs may start running on the node again.

The `drain`, `undrain`, and `drainstatus` commands have the same grouping options. The grouping options can make the command apply to not just one node, but to a list of nodes, a group of nodes, a category of nodes, a rack, a chassis, an overlay, a role, or a status. Continuing the example:

```
[bright92->device]# drain -c default; !# for a category of nodes
Node Queue Status
------------------------ ------------------------ ----------------
node001 PbsPro/pb:workq Drained
node002 UGE/uc/all.q Drained
```

The help text for each command indicates the syntax:

**Example**

```
[root@bright92 ~]# cmsh -c "device help drain"
Name: drain - Drain jobs (not data) on a set of nodes
Usage: drain [OPTIONS/node]

Options: -n, --nodes <node>
  List of nodes, e.g.
  node001..node015,node020..node028,node030 or
  */some/file/containing/hostnames

  -g, --group <group>
  Include all nodes that belong to the node group, e.g.
```
testnodes or test01,test03

-c. --category <category>
Include all nodes that belong to the category, e.g. default or default.gpu

-r. --rack <rack>
Include all nodes that are located in the given rack, e.g rack01 or rack01..rack04

-h. --chassis <chassis>
Include all nodes that are located in the given chassis, e.g chassis01 or chassis03..chassis05

-e. --overlay <overlay>
Include all nodes that are part of the given overlay, e.g overlay1 or overlayA,overlayC

-m. --image <image>
Include all nodes that have the given image, e.g default-image or default-image,gpu-image

-t. --type <type>
Type of devices, e.g node or virtualnode,cloudnode

-i. --intersection
Calculate the intersection of the above selections

-u. --union
Calculate the union of the above selections

-l. --role <role>
Filter all nodes that have the given role

-s. --status <status>
Only run command on nodes with specified status, e.g. UP, "CLOSED|DOWN", "INST.*"

--setactions <actions>
set drain actions, actions already set will be removed (comma-separated)

--appendactions <actions>
append drain actions to already existing drain actions (comma-separated)

--removeactions <actions>
remove drain actions

--listactions
list all drain actions

--clearactions
remove all drain actions
Examples: drain
    drain node001
    drain -r rack01
    drain --setactions reboot

A useful one-liner to reboot and reprovision a sequence of nodes could be in the following format:

Example

cmsh -c 'device; drain -n <nodes> --setactions powerreset ; drain -n <nodes> --appendactions undrain'

This drains the nodes and does a power reset action, which provisions the nodes. After the nodes are up again, they are undrained so that they are ready to accept jobs again. The command allows the sequence of nodes to be rebooted, for example for the purpose of upgrading a kernel, without needing to schedule a downtime for the cluster.

The cluster administrator should be aware that that using the preceding one-liner is not a universal solution. For example, it is not a good solution if the drain takes a long time, because it means that nodes that drain early could be idle for a long time.

7.8 Examples Of Workload Management Assignment

7.8.1 Setting Up A New Category And A New Queue For It

Suppose a new node with processor optimized to handle Shor’s algorithm is added to a cluster that originally has no such nodes. This merits a new category, shornodes, so that administrators can configure more such new nodes efficiently. It also merits a new queue, shorq, so that users are aware that they can submit suitably-optimized jobs to this category.

Ways to configure the cluster manager for this are described next.

A New Category And A New Queue With Bright View For An Existing Workload Manager Cluster

Queues can be added to a WLM cluster by running the WLM wizard (clickpath HPC→WLM Wizard) if the WLM cluster does not yet exist. The wizard provides an option during WLM cluster creation for adding queues.

To create a new queue in an existing WLM cluster, the clickpath HPC→WLM Clusters<WLM cluster name>Job queues→<queue name>→Add is followed. The properties of the new queue can then be set—at least the queue name should be set—in the dialog window of the queue (figure 7.17).
Figure 7.17: Adding A New Queue Via Bright View

The configuration for the queue properties is saved, and then the job queue scheduler configuration is saved.

Next, a new category can be added via the clickpath **Grouping ➔ Node categories ➔ Add**. Parameters in the new category can be set—at least the category name should be set—to suit the new machines (figure 7.18).
The name shornodes can therefore be set here.

Another option within the category is to set the queue. The queue, shorq, is therefore set here for this new category. Setting a queue for the category means configuring the options of the queue scheduler role, for example PBS Pro client role, for the category. Continuing from within the Node categories options window of figure 7.18, the relative clickpath to set a queue is ROLES → Roles → Add → PBS pro client role → Edit → Queues. The appropriate queue can be selected from the queue scheduler menu option (figure 7.19).
In this case, the \textit{shorq} created earlier on in figure 7.17 is presented for selection. After selection, the configuration settings can then be saved—for this example it means going back up through the levels and clicking on the \texttt{Save} button at the node category settings level (the right hand window shown in figure 7.18), and also going up one more level to click the \texttt{Save} button on the node category list window (the left hand window shown in figure 7.18).

Nodes that are to use the queue should be members of the \texttt{shornodes} category. The final step is then to allocate such nodes to the category. This can be done, for example for a node001, by going into the settings of the node, via the clickpath \texttt{Devices $\rightarrow$ Nodes[node001] $\rightarrow$ Edit $\rightarrow$ Settings $\rightarrow$ Category} and setting the category to \texttt{shornodes}.

**A New Category And A New Queue With cmsh**

The preceding example can also be configured in cmsh as follows:

```
[bright92]% wlm; use slrum; jobqueue; add shorq
[bright92->wlm[slrum]->jobqueue[shorq]]% commit
```

The new category, \texttt{shornodes}, can be created by cloning an old category, such as \texttt{default}:

```
[bright92->wlm[slrum]->jobqueue[shorq]]% category
[bright92->category]% clone default shornodes
[bright92->category[shornodes]]% commit
```

Then, going into the roles submode, appropriate workload manager roles can be assigned, and appropriate queues can be appended and committed for that category:

```
[bright92->category[shornodes]]% roles
[bright92->category[shornodes]->roles]% assign slurmclient; commit
[bright92->category[shornodes]->roles[slurmclient]]% append queues shorq
[bright92->category[shornodes]->roles%][commit]
```

The nodes belonging to the \texttt{shornodes} category can then be placed by going into device mode to select the nodes to be placed in that category. For example:
7.8.2 Setting Up A Prejob Check

How It Works
Measurables such as health checks (section 13.2.4) by default run as scheduled tasks over regular intervals. They can optionally be configured to run on demand instead, or as prejob checks, that is, before a job is run.

If a health check is configured with the prejob setting, then its response means the same as that of a health check, that is:

- If the response to a prejob health check is PASS, then it shows that the node is displaying healthy behavior for that particular health check.
- If the response to a prejob health check is FAIL, then it implies that the node is unhealthy, at least for that particular health check.

The reason a prejob health check is treated more carefully is that, if the node it has just run on is unhealthy, then it means that a job submitted to the node may fail, may not be able to start, or may even vanish outright. The way it can vanish in some cases, without any information beyond the job submission “event horizon” leads to this behavior sometimes being called the Black Hole Node Syndrome.

Draining On Prejob Health Check Failure: It can be troublesome for a system administrator to pinpoint the reason for such job failures, since a node may only fail under certain conditions that are hard to reproduce later on. It is therefore a good policy to disallow passing a job to a node which has just been flagged as unhealthy by a health check. Thus, a sensible action (section 13.2.6) taken by a prejob health check on receiving a FAIL response would be to put the node in a Drained state (section 7.7.3). The drain state means that the cluster manager arranges a rescheduling of the job so that the job runs only on nodes that are believed to be healthy. In hindsight it should also be obvious that a node in a drain state can still accept jobs, if the jobs are run from outside of CMDaemon control. Thus, for example if jobs are submitted via workload managers that are not integrated with CMDaemon, then such jobs are not stopped from running on the node.

A node that has been put in a Drained state with a health check is not automatically undrained. The administrator must clear such a state manually.

The failed prejob health check (page 950) is enabled by default, and logs any prejob health check passes and failures. By default there are no prejob health checks configured, so by default this health check should always pass.

Configuration Using cmsh
Prejob health checks have the parameter when set to one of the following values:

1. pre\ job
2. timed and pre\ job

The parameter when for a health check can be set within the monitoring setup mode.

Example
For example, for the ib health check:
[bright92->monitoring->setup]% get ib when
Timed
[bright92->monitoring->setup]% set ib when <TAB><TAB>
on\demand on\start out of\band pre\job timed timed\and\pre\job
[bright92->monitoring->setup]% set ib when pre\job
[bright92->monitoring->setup*]% commit
Configuration Using Bright View
To configure the monitoring of nodes with a prejob check in Bright View, the clickpath:
Monitoring → Data Producers → Monitoring data producers → Edit → When Pre job
is used to set the When value for when it runs. In this case, When is set to Pre job.

Configuration With cm-wlm-setup
To set a prejob health check up via cm-wlm-setup, cm-wlm-setup is run in step-by-step mode. During one of the steps the administrator is prompted to select, with a tick on a checkbox, which of the available health checks are to run as prejob health checks in the cluster manager (figure 7.20):

![Figure 7.20: Prejob healthcheck selection in the cm-wlm-setup screen](image)

If a selected health check was already set as a prejob—that is, with the When parameter already set to either pre job or timed and pre job—then the checkbox is already ticked. If the When parameter was set to timed before, then selecting the health check sets the When parameter to timed and pre job. For all other values of the When parameter for that health check, the When parameter is set to pre job.

Custom Prejob Configuration
For more unusual prejob checking requirements, further details on how prologs are configured are given in section 7.3.4.

7.9 Power Saving With cm-scale
The cm-scale service can be used by an administrator to reduce the energy and storage costs of compute nodes by changing their power state, or their existence state, according to workload demands. That is, cm-scale automatically scales a cluster up or down, on-demand, by powering up physical nodes, cloud nodes, or virtual nodes. The scaling is carried out according to settings in the ScaleServer role which set the nodes that are to use the cm-scale service.

The cm-scale service is covered extensively in Chapter 8.
7.10 Cgroups

Linux system processes and all their future children can be aggregated into sets. These sets can be made into hierarchical groups with specialized behavior using the Control Groups (cgroups) mechanism. The behavior is controlled by different subsystems that are attached to the cgroup. A subsystem may, for example, allow particular CPU cores to be allocated, or it may restrict memory, or it may reduce swap usage by processes that belong to the group, and so on.

Details about Linux cgroups and their subsystems can be found at [https://www.kernel.org/doc/Documentation/cgroups/cgroups.txt](https://www.kernel.org/doc/Documentation/cgroups/cgroups.txt).

As far as workload management is concerned, it makes sense to distinguish between workload manager cgroup settings and system-wide cgroup parameters. The workload manager cgroup settings allow the administrator to configure a workload manager to use cgroups in a particular way, whereas the system-wide cgroup settings allow the administrator to manage cgroups whether a workload manager is used or not.

7.10.1 Cgroups Settings For Workload Managers

If the workload manager allows cgroups usage, then the cluster manager provides capabilities to manage the cgroup parameters within the workload manager.

**Slurm**

Slurm supports 3 cgroups-related plugins. These are all enabled by default, and are:

1. proctrack/cgroup: enables process tracking and suspend/resume capability using cgroups. This plugin is more reliable for tracking and control than the former proctrack/linux.

2. task/cgroup: provides the ability to
   - confine jobs and steps to their allocated cpuset
   - bind tasks to sockets, cores and threads
   - confine jobs and steps to specific memory resources and gres devices

3. jobacct_gather/cgroup: collects accounting statistics for jobs, steps and tasks using the cpuacct, memory, and blkio cgroups subsystems.

Slurm uses 2 configuration files to store the parameters and devices used for cgroup support:

1. /cm/shared/apps/slurm/var/etc/<cluster name>/cgroup.conf: defines parameters used by Slurm’s Linux cgroup-related plugins. The file contains a section that is autogenerated by CMDaemon, with cgroups-related parameters defined in the SlurmServer role.

2. /etc/slurm/cgroup_allowed_devices_file.conf: declares devices that need to be allowed by default for all jobs. The syntax of this file accepts one device per line. It permits wildcard definitions such as /dev/sda* or /dev/cpu/*/*. The path to this file can be changed in cmsh by setting the value of the cgroup parameter Allowed Devices File within the cgroups submode of the WLM mode for the Slurm instance.

**Example**

```
[root@bright92 ~]# cmsh
[bright92]# wlm use slurm
[bright92->wlm[slurm]]# cgroups; get alloweddevicesfile
/etc/slurm/cgroup_allowed_devices_file.conf
```

For Slurm, the administrator can set cgroups parameters using cmsh by going into the cgroups sub-mode of the WLM instance.

Parameters that can be managed include:
<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
<th>Configuration Parameter</th>
</tr>
</thead>
<tbody>
<tr>
<td>Auto Mount</td>
<td>Force Slurm to mount cgroup subsystems if they are not mounted yet</td>
<td>CgroupAutomount</td>
</tr>
<tr>
<td>Mount Point</td>
<td>Where cgroup root is mounted</td>
<td>CgroupMountpoint</td>
</tr>
<tr>
<td>Task Affinity</td>
<td>Set a default task affinity to bind each step task to a subset of the allocated cores using sched.setaffinity</td>
<td>TaskAffinity</td>
</tr>
<tr>
<td>Release Agent Dir</td>
<td>Directory containing Slurm cgroup release_agent files</td>
<td>CgroupReleaseAgentDir</td>
</tr>
<tr>
<td>Allowed Devices File</td>
<td>File used to declare the default devices for all the jobs, if ConstrainDevices is true</td>
<td>AllowedDevicesFile</td>
</tr>
<tr>
<td>Constrain Cores</td>
<td>Constrain allowed cores to the subset of allocated resources</td>
<td>ConstrainCores</td>
</tr>
<tr>
<td>Constrain RAM Space</td>
<td>Constrain the job’s RAM usage</td>
<td>ConstrainRAMSpace</td>
</tr>
<tr>
<td>Constrain Swap Space</td>
<td>Constrain the job’s swap space usage</td>
<td>ConstrainSwapSpace</td>
</tr>
<tr>
<td>Constrain Devices</td>
<td>Constrain the job’s allowed devices (also constrain GPUs) based on GRES allocated resources</td>
<td>ConstrainDevices</td>
</tr>
<tr>
<td>Allowed RAM Space</td>
<td>Percentage memory (default is 100%) out of the allocated RAM allowed for the job cgroup RAM. If this percentage is exceeded, then the job steps will be killed and a warning message will be written to standard error.</td>
<td>AllowedRAMSpace</td>
</tr>
<tr>
<td>Allowed Swap Space</td>
<td>Percent allocated memory allowed for the job cgroup swap space</td>
<td>AllowedSwapSpace</td>
</tr>
</tbody>
</table>

...continues
...continued

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
<th>Configuration Parameter</th>
</tr>
</thead>
<tbody>
<tr>
<td>Max RAM Percent</td>
<td>Maximum percent of total RAM for a job</td>
<td>MaxRAMPercent</td>
</tr>
<tr>
<td>Max Swap Percent</td>
<td>Maximum percent of total RAM for the amount of RAM+Swap that may be used for a job</td>
<td>MaxSwapPercent</td>
</tr>
<tr>
<td>Min RAM Space</td>
<td>Minimum MB for the memory limits defined by Allowed RAM Space and Allowed Swap Space</td>
<td>MinRAMSpace</td>
</tr>
</tbody>
</table>

* Boolean (takes yes or no as a value)

The options are always written in the cgroup.conf file. More details on these options can be found in the man page man(5) cgroup.conf.

**UGE**

Univa Grid Engine allows cgroups settings to be defined at two levels.

- Globally: The parameters can be set globally in the global configuration. The cluster manager defines cgroups global settings in the cgroups submode of the UGEServer role.

- Per host: The parameters can be overridden in the host configuration for particular hosts. The cluster manager defines cgroups host settings in the cgroups submode of the UGEClien role.

The global and host level cgroups parameters are the same, and are as follows:

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
<th>Configuration Parameter</th>
</tr>
</thead>
<tbody>
<tr>
<td>Auto Mount*</td>
<td>Force UGE to mount a cgroup subsystems if they are not mounted yet</td>
<td>mount</td>
</tr>
<tr>
<td>Cpuset</td>
<td>If true, then core binding is done by the cgroup cpuset subsystem</td>
<td>cpuset</td>
</tr>
</tbody>
</table>

...continues
<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
<th>Configuration Parameter</th>
</tr>
</thead>
<tbody>
<tr>
<td>Freezer</td>
<td>If true, then it enables the cgroup freezer subsystem for job suspension and re-sumption</td>
<td>freezer</td>
</tr>
<tr>
<td>Freeze PE Tasks</td>
<td>If true and the freezer subsystem is turned on, then the master task is suspended, and all slave tasks of the parallel job are also frozen</td>
<td>freeze_pe_tasks</td>
</tr>
<tr>
<td>Killing</td>
<td>If true then UGE signals all processes forked/started by the job until all of them are killed</td>
<td>killing</td>
</tr>
<tr>
<td>Forced NUMA</td>
<td>If true then on NUMA machines only local memory (memory in the same NUMA zone) is allowed to be used when the job requested memory allocation with -s bind cores:strict</td>
<td>forced_numa</td>
</tr>
<tr>
<td>Virtual Memory</td>
<td>Specifies if virtual memory can be limited with cgroups</td>
<td>h_vmem_limit</td>
</tr>
<tr>
<td>Limit</td>
<td></td>
<td>h_mem_free_hard</td>
</tr>
<tr>
<td>Memory Free Hard</td>
<td>If true then kernel ensures that the job does not use more main memory than required</td>
<td>h_mem_free_soft</td>
</tr>
<tr>
<td>Memory Free Soft</td>
<td>If true, and the hard memory limit is turned off, then the requested memory with m_mem_free is a soft limit</td>
<td>...continues</td>
</tr>
</tbody>
</table>
...continued

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
<th>Configuration Parameter</th>
</tr>
</thead>
<tbody>
<tr>
<td>Min Memory Limit</td>
<td>A host based minimum memory limit, in bytes or values like 10M, 1G</td>
<td>min_memory_limit</td>
</tr>
<tr>
<td>Devices</td>
<td>Controls which devices the job can access. It permits wildcard definitions such as /dev/sda* or /dev/cpu/*/</td>
<td></td>
</tr>
<tr>
<td>Options</td>
<td>Additional cgroups parameters for UGE</td>
<td></td>
</tr>
</tbody>
</table>

* Boolean (takes yes or no as a value)

The options are updated in UGE only when cgroups are enabled in the UGEServer or UGEClient roles. To enable them, enabled must be set to true within the cgroups submode of the role. By default only global cgroups settings are enabled in UGE.

**PBS**

PBS supports cgroups through a special Python hook that is installed by cm-wlm-setup by default. The hook name is pbs_cgroups.

PBS cgroup options can be set managed cgroups in cmsh or Bright View. For example, in cmsh the options can be shown with:

**Example**

```
[bright92->wlm[openpbs]->cgroups]% show
Parameter                Value
---------------------------------------------------------------
Mount Point              /sys/fs/cgroup
Revision                 
Type                     PbsProCgroupsSettings
...                      
Nemsw reserve amount     64MiB
```

The following table shows the more important cgroup options that can be set:

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
<th>Default Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>Mount Point</td>
<td>Where cgroup root is mounted</td>
<td>/sys/fs/cgroup</td>
</tr>
<tr>
<td>Revision</td>
<td>Entity revision</td>
<td></td>
</tr>
<tr>
<td>Type</td>
<td>Type of entity</td>
<td>PbsProCgroupsSettings</td>
</tr>
<tr>
<td>Job cgroup template</td>
<td>Template for job cgroup path</td>
<td>pbspro.slice/ (ESCAPE_JOBID will be replaced by pbspro-$ESCAPE_JOBID.slice systemd-escape of job id)</td>
</tr>
</tbody>
</table>

...continues
### 7.10 Cgroups

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
<th>Default Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>Cgroup prefix</td>
<td>Cgroup prefix used by PBS when the cgroup is created</td>
<td>pbspro</td>
</tr>
<tr>
<td>Auto Mount *</td>
<td>If true then workload manager tries to mount a subsystem if it is not mounted yet</td>
<td>no</td>
</tr>
<tr>
<td>Enabled *</td>
<td>When set the cgroups hook is enabled (in the hook config: enabled)</td>
<td>yes</td>
</tr>
<tr>
<td>Nvidia SMI</td>
<td>The location of the nvidia-smi command (in the hook config: nvidia-smi)</td>
<td>/usr/bin/nvidia-smi</td>
</tr>
<tr>
<td>Kill timeout</td>
<td>Maximum number of seconds the hook spends attempting to kill job processes before destroying cgroups (in the hook config: kill_timeout)</td>
<td>10s</td>
</tr>
<tr>
<td>Server timeout</td>
<td>Maximum number of seconds the hook spends attempting to fetch node info from the server (in the hook config: server_timeout)</td>
<td>15s</td>
</tr>
<tr>
<td>Use hyperthreads *</td>
<td>All CPU threads are made available to jobs (in the hook config: use_hyperthreads)</td>
<td>no</td>
</tr>
<tr>
<td>Ncpus are cores *</td>
<td>Ncpus of a vnode is the number of cores, and the hook assigns all threads of each core to a job (in the hook config: ncpus_are_cores)</td>
<td>no</td>
</tr>
<tr>
<td>Cpuacct enabled *</td>
<td>Enable cpuacct cgroup controller for jobs</td>
<td>yes</td>
</tr>
<tr>
<td>Cpuset enabled *</td>
<td>Enable cpuset cgroup controller for jobs</td>
<td>yes</td>
</tr>
<tr>
<td>Devices enabled *</td>
<td>Enable devices cgroup controller for jobs</td>
<td>yes</td>
</tr>
<tr>
<td>Devices allow</td>
<td>Parameter specifies how access to devices will be controlled</td>
<td>b <em>:</em> rwm,c <em>:</em> rwm</td>
</tr>
<tr>
<td>Hugetlb enabled *</td>
<td>Enable hugetlb cgroup controller for jobs</td>
<td>no</td>
</tr>
<tr>
<td>Hugetlb default</td>
<td>The amount of huge page memory assigned to the cgroup when the job does not request hpmem</td>
<td>0B</td>
</tr>
<tr>
<td>Hugetlb reserve percent</td>
<td>The percentage of available huge page memory (hpmem) that is not to be assigned to jobs</td>
<td>0</td>
</tr>
<tr>
<td>Hugetlb reserve amount</td>
<td>An amount of available huge page memory (hpmem) that is not to be assigned to job</td>
<td>0B</td>
</tr>
<tr>
<td>Memory enabled *</td>
<td>Enable memory cgroup controller for jobs</td>
<td>yes</td>
</tr>
</tbody>
</table>

...continues
Parameter | Description | Default Value
---|---|---
Memory soft limit<sup>*</sup> | If false, then PBS uses hard memory limits which prevent the processes from ever exceeding their requested memory usage | yes
Memory default | Amount of memory assigned to the job if it doesn’t request any memory | 64MiB
Memory reserve percent | The percentage of available physical memory that is not to be assigned to jobs | 0
Memory reserve amount | A specific amount of available physical memory that is not to be assigned to jobs | 64MiB
Memsw enabled<sup>*</sup> | Enable memsw cgroup controller for jobs | no
Memsw default | Specifies the amount of memory + swap assigned to the job if it doesn’t request any memory | 256MiB
Memsw reserve percent | Percentage of available swap that is not to be assigned to jobs | 0
Memsw reserve amount | An amount of available swap that is not to be assigned to jobs | 64MiB

* Boolean (takes yes or no as a value)

**LSF**

LSF allows resource enforcement to be controlled with the Linux cgroup memory and cpuset subsystems. By default, when LSF is set up with cm-wlm-setup, then both subsystems are enabled for LSF jobs. If job processes on a host use more memory than the defined limit, then the job is immediately killed by the Linux cgroup memory subsystem. The cgroups-related configuration options are available in cmsh or Bright View, and can be found in the cgroups submode of the LSF cluster settings:

Parameter | Description | Configuration Parameter
---|---|---
Resource Enforce | Controls resource enforcement through the Linux cgroups memory and cpuset subsystem, on Linux systems with cgroups support. The resource can be either memory or cpu, or both cpu and memory, in either order (default: memory cpu) | LSB_RESOURCE_ENFORCE

...continues
7.11 Custom Node Parameters

NVIDIA Bright Cluster Manager 9.2 allows the administrator to specify the most important node parameters when a node is configured in a workload manager. But sometimes the workload manager allows the configuration of other advanced or user-defined settings for the nodes. When an administrator clones a node in the cluster manager, or creates it from scratch, then those advanced settings may need to be configured manually per node in the workload manager. Sometimes the nodes are created or cloned automatically, for example cm-scale (Chapter 8), in which case node parameters customization is not possible without manual intervention in the workload manager configuration.

Since NVIDIA Bright Cluster Manager version 8.2, the administrator can configure the advanced node settings in the node customizations mode of cmsh or Bright View. The customizations are available in the workload manager client roles, so they can be applied at the node, node category or configuration overlay levels.

In order to configure a custom node parameter for the workload manager, the administrator adds the node customization entry and sets its value. The entry is enabled by default, using the parameter enabled. Setting the value to disabled automatically removes the parameters from the node settings of the workload manager.

For example, in order to add a new customization entry for PBS, the following cmsh commands can be used:

**Example**

```
[bright92]% category use default
[bright92->category[default]]% use pbsproclient
...->roles[pbsproclient]]% nodecustomizations
...->roles[pbsproclient]->nodecustomizations]% add resv_enable
...->roles[pbsproclient]->nodecustomizations*[resv_enable*]]% set value False
...->roles[pbsproclient]->nodecustomizations*[resv_enable*]]% show
```

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
<th>Configuration Parameter</th>
</tr>
</thead>
<tbody>
<tr>
<td>Process Tracking*</td>
<td>This parameter, when enabled, has LSF track processes based on job control functions such as termination, suspension, resume, and other signals, on Linux systems which support the cgroups freezer subsystem</td>
<td>LSF_PROCESS_TRACKING</td>
</tr>
<tr>
<td>Linux Cgroup Accounting*</td>
<td>When enabled, processes are tracked based on CPU and memory accounting for Linux systems that support cgroup’s memory and cpacct subsystems</td>
<td>LSF_LINUX_CGROUP_AOCT</td>
</tr>
</tbody>
</table>

* Boolean (takes yes or no as a value)
Different workload managers allow different kinds of node settings to be set. For example, for the Grid Engines, the settings are the so-called complex attributes, as defined by the administrator. But for PBS, the node settings are built-in, and so the administrator can only change their values but not their names. So when the administrator configures node customization entries, the cluster manager configures them differently depending on the workload manager that is used.

Thus if the administrator creates a new customization entry with key name <KEY> and value <VALUE>, then this is applied to the workload managers as follows:

• **Slurm.** The node customization entry is appended to the NodeName line of the particular nodes in `slurm.conf` in the form `<KEY>=<VALUE>`. If the entry is removed or disabled in the cluster manager configuration, then the entry `<KEY>=<VALUE>` pair is removed from the NodeName line.

• **PBS Pro, OpenPBS and PBS Pro CE.** The entry is configured with the `qmgr` utility, using this command: `set node NODENAME <KEY>=<VALUE>`. If the entry is removed or disabled, then it is unset in the `qmgr`.

• **UGE.** The entry is applied to an exechost complex attribute that must already be defined in the configuration. The complex attribute name is set to `<KEY>` and its value to `<VALUE>`. If the entry is removed or disabled, then it is removed from the exechost settings of the node.

• **LSF.** The entry is set in the `lsb.hosts` file, in the `Hosts` section. If the `<KEY>` of the entry is defined as one of the section columns, then the parameter value is replaced by the cluster manager to `<VALUE>`. When the customization entry is removed or disabled, then its value is replaced by () in the section.
NVIDIA Bright Cluster Manager
Auto Scaler

8.1 Introduction

NVIDIA Bright Cluster Manager Auto Scaler can be used by an administrator to reduce the energy and storage costs of compute nodes by changing their power state, or their existence state, according to workload demands. The idea behind Auto Scaler is that it automatically scales a cluster up or down, on-demand, by powering up physical nodes or cloud nodes.

On the cluster itself, Auto Scaler is implemented by the `cm-scale` service. The scaling is carried out according to settings in the `ScaleServer` role which set the nodes that are to use the `cm-scale` service.

The `cm-scale` service runs as a daemon. It collects information about workloads from different workload engines, and it uses knowledge of the nodes in the cluster. In the case of HPC jobs, the daemon also gathers knowledge of the queues that the jobs are to be assigned to, and also gathers knowledge on which of the HPC jobs are requesting exclusive node access.

Based on the workload engine information and queues knowledge, the `cm-scale` service can clone and start compute nodes when the workloads are ready to start. The service also stops or terminates compute nodes, when no queued or running workloads remain on the managed nodes.

8.1.1 Use Cases

The `cm-scale` service can be considered as a basis for the construction of different kinds of dynamic data centers. Within such centers, nodes can be automatically re-purposed from one workload engine setup to another, or they can be powered on and off based on the current workload demand.

A few use cases are discussed next, to show how this basis can be built upon:

1. An organization wants to run PBS Pro and Slurm on the same cluster, but how much one workload engine is used relative to the other varies over time. For this case, nodes can be placed in the same pool and shared. When a node finishes an existing job, the `cm-scale` service can then re-purpose that node from one node category to another if needed, pretty much immediately. The re-purposing decision for the node is based on the jobs situation in the PBS Pro and Slurm queues.

2. An organization would like to use their cluster for both Kubernetes and for Slurm jobs. For this case, the admin adds Kubernetes- and Slurm-related settings to the `ScaleServer` role. Using these settings, the `cm-scale` service then switches nodes from one configuration overlay to another. For example, if Slurm jobs are pending and there is a free Kubernetes node, then the node is turned into a Slurm node pretty much immediately.

---

1 HPC UGE jobs have no queue assignment by default, and are therefore ignored by default. Adding a line such as:

```
-q all.q
```

To the `var/<instance name>/common/sge_request` file, under `/cm/shared/apps/uge/`, assigns UGE jobs to the `all.q` queue by default, so that `cm-scale` considers them.
3. An organization has a small cluster with Slurm installed on it, and would like to be able to run more jobs on it. This can be carried out using the cluster manager’s Cluster Extension cloudbursting capability (Chapter 3 of the Cloudbursting Manual) to extend the cluster when there are too many jobs for the local cluster. The extension to the cluster is made to a public cloud provider, such as AWS or Azure. For this case, the users can use the cmjob utility (section 4.7 of the User Manual, and section 4.3 of the Cloudbursting Manual) to submit their cloud jobs, and Slurm sbatch for their regular jobs. The cm-scale service then tracks the Slurm queues, and decides whether or not new cloud nodes should be added from the cloud as extensions to the local network and queues. When the jobs are finished, then the cloud nodes are terminated automatically. Cluster extension typically takes several minutes from prepared images, and longer if starting up from scratch, but in any case this change takes longer than simple re-purposing does.

4. An organization runs PBS Pro only and would like to power down free, local, physical nodes automatically, and start up such nodes when new jobs come in, if needed. In this case, cm-scale follows the queues and decides to stop or to start the nodes automatically depending on workload demand. Nodes typically power up in several minutes, so this change takes longer than simple re-purposing does.

8.1.2 Resource Constraints

1. Two workload engines of the same kind cannot be served by cm-scale on the same cluster. For example, two Slurm instances cannot be managed, but Slurm and LSF can.

2. When the service considers whether or not the node is suited for the workload, it considers the following requested resources:
   
   (a) the number of CPU cores;
   
   (b) the number and type of GPUs;
   
   (c) the amount of memory.

   It does not consider other types of resources. For example, if a workload needs Xeon Phis, then cm-scale does not verify if the node actually includes Xeon Phis. The validation is however performed by the engine. If a resource requested by the workload cannot be provided by the node, then the engine (the workload manager) specifies a pending reason that is used by cm-scale to decide on node operations.

   In cmsh, the wlmresources command displays the resources that Auto Scaler considers.

Example

```bash
[root@bright92 ~]# cmsh
[bright92]# device
[bright92->device]# wlmresources
WLM Name Amount Nodes
-------- ----------------- -------------- ----------------
pbspro cpu_total 32 node001..node006
pbspro gpu_free 1 node005
pbspro gpu_free 2 node001..node00+
pbspro mem_free 8,108,032,000 node004,node005
pbspro mem_free 948,224,000 node006
slurm cpu_alloc 2 node001..node006
slurm cpu_total 16 node006
slurm cpu_total 8 node001..node005
slurm gpu_free 1 node005
slurm gpu_free 2 node001..node004
slurm mem_free 1,000,000,000 node006
```
These resources are taken from appropriate workload manager, and not necessarily equal to the available physical resources on the nodes.

**Number Of CPU Cores Calculation**

The cm-scale service uses the number of CPU cores on a node in order to map workloads to the node. The calculation of this number is optimized for many node scenarios.

For each node, at the beginning of each cm-scale iteration, the following procedure is followed step-by-step to determine the CPU cores number. The procedure for the current iteration stops when a step yields a non-zero value, otherwise the next step is tried.

1. If the engine is a workload manager, then the workload manager client role is considered and its slots parameter is used. In the very unusual case of several workload manager roles assigned to the node at the same time, then the minimum number of slots is calculated.

2. If the node is a cloud node, then its flavor Type parameter in the case of EC2, or VMSize in case of Azure, is used. These parameters are accessible via cmsh, within device mode for the cloud node, within the cloudsettings submode. For EC2 nodes the flavor value (Type) can be set to a long statement such as: "62 EC2 Compute Units (16 virtual cores with 3.00 EC2 Compute Units each)", and for this statement 16 is used as the number of CPU cores. If the flavor is not defined on the node, then its cloud provider is considered.

3. If a template node is defined in the dynamic node provider, and it is a cloud node, then the template node flavor is used for the CPU core number in the same way as shown in the preceding step.

4. If the Default Resources parameter (in a resource provider) contains "cpus=N", where N is a number of CPU cores, then N is used.

5. If the node exists at this moment (for example, it is not just cloned in the cluster manager), then the CPU cores number is requested from CMDaemon, which collects nodes system information. This is used as one of the last methods because it is slow.

6. If the node is defined in the ScaleServer role via the dynamic resource provider, then the CPU cores number is retrieved from its template node. This method is as slow as the preceding method. It is therefore not recommended when many nodes are managed by cm-scale, otherwise the iteration can take several minutes. In such a case, setting the slots parameter manually is typically wiser, so that step 1 is the step that decides the CPU cores number.

If cm-scale does not manage to find a value for the node, then it prints a warning in the log file and does not make use of the node during the iteration. It is usually wise for the administrator to set the slots parameter manually for nodes that otherwise persistently stay unused, so that resources are not wasted.
Requested GPUs
When a user of a workload manager requests a number of GPUs for the job that is to be run, then Auto Scaler maps this job to nodes that have enough available GPUs. In the case of Slurm, in addition to the number of GPUs, Auto Scaler recognizes the GPU type if the user specifies it. For other workload managers only number of GPUs is counted.

In order to find out how many GPUs and what types are available, Auto Scaler, gets this information from a workload manager via CMDaemon. Thus if some GPUs are in use, or if the administrator configures fewer GPUs than the node has, then Auto Scaler still knows what GPUs the workload managers can use during their job scheduling. The number and type of requested GPUs, as well as the number and types of available GPUs, can be found in the log file /var/log/cm-scale.log if debug messages are enabled in ScaleServer role.

Requested Memory
A memory request is considered by Auto Scaler if a user specifies this request when the job is submitted. If the user does not specify the memory, then a value of 0 is assumed.

UGE does not provide the amount of memory requested by a user per node, but allocates memory per CPU core. Thus, Auto Scaler by default operates with a memory amount per CPU core, which can also be seen in the logs.

Default Resources Specification
Sometimes the available consumable resources must be defined explicitly by administrator. This is needed in the case of LSF, because when a node is down, LSF does not provide the available consumable resources configured for the node. Therefore in this case Auto Scaler does not know if the node actually has any of the resources known to LSF. The mechanism used for defining explicitly can also be used for other workload managers, for testing purposes.

For now, only the following types of consumable resources can be specified as default resources to the Default Resources setting, under the Resource Provider parameter. The setting is a list of strings, where each string specifies one of the following resources:

1. cpu: the number of CPU cores that will be used, if other sources for this information do not provide a value (the calculation for the number of CPU cores is described on page 369);

2. mem_free: the amount of available memory. If no units are specified, then bytes are assumed. It is also possible to append one of the following units:
   - KB or K (kilobytes), KIB (kibibytes)
   - MB or M (megabytes), MiB (mebibytes)
   - GB or G (gigabytes), GiB (gigibytes)
   - TB or T (terabytes), TiB (tebibytes)
   - PT or P (petabytes), PiB (pebibytes)

The format of the memory specification is the following:

mem_free:<engine>=<amount>

where <engine> is the name of the workload engine which "provides" this value, and <amount> is the amount of memory.

Example

mem_free:lsf1=32GB
3. **gpu_free**: the number of available GPUs. The format of the GPUs specification is:

\[
gpu\_free[:<\text{type}>]:<\text{engine}>=<\text{number}>
\]

where \(<\text{type}>\) is a string that specifies the GPUs type (available only for Slurm). Only a single GPU type per node is currently supported. \(<\text{number}>\) is a number of GPUs, and \(<\text{engine}>\) is the name of the workload engine which "provides" this value.

**Example**

\[
gpu\_free:a100:uge3=8
\]

or

\[
gpu\_free:lsf=1
\]

### 8.1.3 Setup

In order to set up Auto Scaler the administrator can run the `cm-auto-scaler-setup` script. The setup allows one of the three base scenarios to be configured. This is possible in express mode as well as in step-by-step mode. When the setup is complete, the administrator can further tune the behaviour of Auto Scaler within the ScaleServer role. The ScaleServer role is always assigned to the head nodes, via a new configuration overlay named autoscaler.

The setup tool assigns the role. The role, in turn, starts the `cm-scale` system service (Auto Scaler). Auto Scaler can be disabled by running `cm-auto-scaler-setup` again, and selecting the menu item Disable. When Auto Scaler is disabled, the configuration overlay is removed.

The service writes logs to `/var/log/cm-scale.log`. Running the `cm-scale` daemon in the foreground for debugging or demonstration purposes is also possible, using the `-f` option. Other additional options that may be used, including at the same time, are:

- `-d`: debug logs
- `-i <N>`: number (<N>) of iterations

The logs are then duplicated to STDOUT:

**Example**

```
root@bright92$ module load cm-scale
root@bright92$ cm-scale -f
[...]
```

When `cm-auto-scaler-setup` is started the administrator can select from one of the following setup operations (figure 8.1):

- express setup,
- step-by-step setup,
- disable.
Express setup allows an initial setup to be carried out, without many questions being asked. It applies default values wherever possible. Some inputs that do not have default values are still needed. Express setup is a good start for the administrators who have never used Auto Scaler before.

Step-by-step setup allows an initial setup to be carried out too, but with some more questions to help tune the cluster to the needs of the administrator. This is more suitable for administrators with some experience in Auto Scaler configuration.

After the wizard has carried out the deployment, both the express and the step-by-step configuration can have their configurations tuned further via `cmsh`, within the `scaleserver` role. Express setup and step-by-step setup both allow one of the 3 pre-defined use-case scenarios to be selected (figure 8.2):

1. **Workload Manager (On-premises)**: Auto Scaler tracks selected workload manager queues, and starts or stops specified on-premises nodes on demand. A static node provider is automatically added to the role for this scenario.

2. **Workload Manager (Cluster extension)**: Auto Scaler dynamically clones nodes from a cloud template node, on demand. The nodes can be terminated or stopped when idle. This scenario is used when cloudbursting is set up. A dynamic node provider is added automatically to the role for this scenario.

3. **Kubernetes (On-premises)**: Auto Scaler tracks Kubernetes jobs or individual pods, and starts or stops the nodes on demand. A static node provider is automatically added to the role for this scenario.
Static nodes and dynamic nodes providers are discussed in section 8.2.2.

In step-by-step mode, for the next step, the dialog suggests specifying Auto Scaler base options (figure 8.3):

![Figure 8.3: Auto Scaler Setup Base Options](image)

The Auto Scaler base options are:

- **Enable debug messages**: Auto Scaler adds debug messages to its log file (default: `/var/log/cm-scale`).
- **Dry run mode**: Disables actual execution of any operation on the cluster. Auto Scaler decisions are still all written to the log file.
- **Run interval**: Number of seconds that Auto Scaler waits before making new decisions regarding cluster auto scaling.

**Use Case: Workload Manager (On-premises)**

When this scenario is selected, then the next step is to configure the static nodes provider. First, categories and individual nodes to be managed by Auto Scaler are selected. The nodes of a selected category, or individual nodes, are added to the node provider (figures 8.4 and 8.5):

![Figure 8.4: Auto Scaler Setup Categories Selection For Static Node Provider](image)
The next step is to pick the workload manager cluster (figure 8.6):

If only one workload manager instance exists, then the screen is skipped. The screen provides a list of workload manager names that \texttt{cm-wlm-setup} has set up. The \texttt{cm-auto-scaler-setup} wizard only configures one workload manager cluster, but others can be added later as separate workload engines with the \texttt{ScaleServer} role.

If running express mode, then the summary screen is displayed. The summary screen includes options to just show the configuration file, or to save the configuration and deploy the setup (figure 8.7):

If running step-by-step mode, then there are some additional tune up screens:
Values can be set for resources in the default node resources screen (figure 8.8):

1. Number of CPU cores per node. Format: `<number>`

2. Number (and optionally, type) of GPUs per node. Format: `[:<type>]:<number>`

3. Available memory for jobs per node. Units can be specified as: KB, K, MiB, MB, M, GiB, GB, G, TiB, TB, T, PiB, PT, P. Format: `<amount>[<unit>]`. If no units are specified, then bytes are assumed.

When Auto Scaler considers whether or not the node is suited for the workload, it considers the following requested resources:

In `cmsh`, the `wlmresources` command, executed in `devices` mode, displays the resources that Auto Scaler considers. These resources are taken from the corresponding workload manager, and are not necessarily equal to the available physical resources on the nodes. Sometimes the available consumable resources must be defined explicitly by the administrator. This is needed if the node never started, or if the WLM (such as in the case of LSF) does not provide node resource information when the node is down. It is recommended that these values are always defined. If nodes vary in their resource requirements, then, after setup, an administrator can add new resource providers (within the ScaleServer role) and set the various default resources for the various groups of nodes.

In the next screen the administrator can specify some engine settings (figure 8.9):

The settings are:

- **Engine Priority**: The workload engine priority. This value is used when the final (global) workload priority is calculated by Auto Scaler. If 0, then this priority is not taken into account.

- **Workloads Per Node**: The maximum number of jobs that can be started on a node in parallel.

Auto Scaler fetches the workload priority values from the settings specified in the next screen (figure 8.10):
The settings are:

- **Fetched**: Priorities are fetched from workload engine. Age and engine priorities are ignored. This option sets the age factor to 0.0, and engine priority to 0.

- **Calculated**: Priorities are calculated from the workload age and engine priority. Both can be tuned by the administrator in ScaleServer role. This option sets the age factor to 1.0, and the external priority factor to 0.0.

Workload trackers settings can be set in the next screen (figure 8.11):

The settings are:

- **Queue Length Threshold**: Number of pending workloads. If this number is reached, then nodes are triggered to start up.

- **Age Threshold**: Workload pending time threshold, in seconds. If a workload reaches this age while pending, then nodes are triggered to start up for that workload.

- **Workloads Per Node**: Number of jobs that can be scheduled to the same node at the same time. A value of 0 means no limit is set.

The settings are applied to all configuring queue trackers. The values can be tuned further afterwards in the ScaleServer role.

**Use Case: Workload Manager (Cluster Extension)**

In the case of workload manager cluster extension scenario, a dynamic node provider with a template node is configured. In this scenario the administrator should expect cloud nodes to be triggered, which run in a previously configured and deployed cluster extension.
Cluster extension configuration and deployment in the cluster manager is described in Chapter 3 of the *Cloudbursting Manual*, and can be carried out, for example, for a particular cloud provider. For command line deployment, the `cm-cluster-extension` setup script can be run.

Cloud nodes are thus cluster nodes that extend into a cloud provider, and which are cloned and terminated depending on workload demand.

To configure a cluster extension with Auto Scaler, the administrator is asked to pick a cloud provider (figure 8.12):

![Auto Scaler Setup Cloud Provider](image)

Figure 8.12: Auto Scaler Setup Cloud Provider

With the express setup, the next screen asks for workload manager selection (figure 8.6). With the step-by-step setup, however, some additional screens are presented before getting to the workload manager selection screen. These extra screens are described next.

The Auto Scaler template node selection screen (figure 8.13) prompts for the selection of a template node that is to be used for cloud node cloning.

![Auto Scaler Setup Template Node](image)

Figure 8.13: Auto Scaler Setup Template Node

The selected template node is then set in the dynamic node provider.

The Auto Scaler incrementing network interface screen (figure 8.14) prompts the administrator to select the network interface on the template node that is automatically increment when the node is cloned.
The Auto Scaler node range specification screen (figure 8.15) prompts the administrator to specify a node range. Range format can be used. Nodes are automatically created by Auto Scaler on demand in the cloud according to the range specified.

The remaining screens in this use case have been covered in the earlier section (Use Case: Workload Manager (On-premises)), and are:

- default node resources (figure 8.8),
- engine settings (figure 8.9),
- workload priorities (figure 8.10),
- tracker settings (figure 8.11).

Use Case: Kubernetes (On-premises)
Selection of the Kubernetes (On-premises) scenario leads to a few screens related to Kubernetes and Auto Scaler integration. Auto Scaler tracks Kubernetes jobs or individual pods, and can start or stop the on-premises nodes on demand.

The first screen displayed after the scenario is selected, is a screen that asks for node categories and individual nodes that are to be configured in the static resource provider. Such nodes are the only ones to be managed by Auto Scaler. Category and node selection screens are then displayed as in the earlier sections (figures 8.4 and 8.5).

The administrator is then prompted to pick a Kubernetes cluster (figure 8.16).
If there is only one Kubernetes cluster, then this screen is skipped, and the Kubernetes configuration is used for the integration with Auto Scaler.

The next screen prompts for the Kubernetes engine settings (figure 8.17):

- **Engine Priority**: A workload engine priority. This value is used when the final (global) workload priority is calculated by Auto Scaler. If 0, then this priority is not taken into account.

- **Workloads Per Node**: The maximum number of workloads that can be started on a node.

- **CPU Busy Threshold**: The CPU load % that defines if node is too busy for new pods.

- **Memory Busy Threshold**: The Memory load % that defines if node is too busy for new pods.

The administrator is then prompted to pick the Kubernetes namespace that will be tracked by Auto Scaler (figure 8.18):
In step-by-step mode, the namespace tracker settings are then displayed (figure 8.18):

The settings are:

- **Queue Length Threshold**: If this number of pending workloads is exceeded, then that triggers nodes starting up.
- **Age Threshold**: Workload pending time threshold, in seconds. If the age of the workload is greater than this, then nodes are triggered to start for this workload.
- **Workloads Per Node**: Number of workloads that can be scheduled to the same node at the same time (0 means no limit).

The settings are applied to all configuring namespace trackers. The values can be tuned further afterwards within the ScaleServer role.

The summary screen is displayed next (figure 8.7). Selecting **Save config & deploy** saves the configuration and starts the setup procedure.

### 8.1.4 Comparison With cm-scale-cluster

Versions of NVIDIA Bright Cluster Manager prior to 8.0 had the cm-scale-cluster utility, which provided similar functionality to the current cm-scale utility. However, the older utility did not allow workloads (jobs) to be considered for more than one workload manager at a time, and also supported only HPC workload managers as workload engines.

The current cm-scale service on the other hand, allows, for example, Slurm workloads to be considered on the same cluster, and also supports other types of workload engines besides HPC workload managers.
Furthermore, the cm-scale utility does not use a crontab entry. The old cm-scale-cluster utility required cron to run it regularly, because only one iteration of decision-making was performed by it. The new cm-scale utility, on the other hand, runs the decision-making loop internally within the cm-scale daemon. The decision-making interval is set in the ScaleServer role.

8.2 Configuration

8.2.1 The ScaleServer Role

To configure cm-scale, the cluster administrator configures the ScaleServer role. The role is typically assigned to head nodes:

Example

```
[bright92]% device use master
[bright92->device[bright92]]% roles
[bright92->device[bright92]->roles]% assign scaleserver
```

The role is configured by setting values to its settings. There are some advanced settings for less common options:

```
[bright92->device[bright92]->roles[scaleserver]]% show
Parameter                        Value
--------------------------------- ----------------------------------------------
Add services                     yes
Name                              scaleserver
Provisioning associations        <0 internally used>
Type                             ScaleServerRole
Engines                          <0 in submode>
Resource Providers               <0 in submode>
Dry Run                          no
Debug                            no
Run Interval                     120
Advanced Settings                <submode>
```

```
[bright92->device[bright92]->roles[scaleserver]]% advancedsettings
[bright92->device[bright92]->roles[scaleserver]->advancedsettings]% show
Parameter                        Value
--------------------------------- ----------------------------------------------
Debug2                           no
Max Threads                      16
Power Operation Timeout          30
Connection Retry Interval        5
Log File                         /var/log/cm-scale.log
Pin Queues                       no
Mix Locations                    yes
Failed Node Is Healthy           no
Azure Disk Image Name            images
Azure Disk Container Name        vhds
Azure Disk Account Prefix        
Node Selection                   Alphabetically
Node Selection Uptime Period     2w
```

The advanced settings are:

- **Debug2**: Enable printing very low level debug messages in the log file. This setting must be used with caution because it leads to a rapid increase in the log file size.
- **Max Threads**: Maximum number of threads for sequential RPCs to CMDaemon.
• **Power Operation Timeout**: Power operation RPC timeout, in seconds.

• **Connection Retry Interval**: Connection to CMDaemon retry interval, in seconds.

• **Log File**: Path to the log file (Default `/var/log/cm-scale.log`).

• **Pin Queues**: Pin workloads to their queue nodes.

• **Mix Locations**: Allow workload to be mapped to different locations (cloud and local).

• **Failed Node Is Healthy**: Do not start a new node instead of a failed one.

• **Azure Disk Image Name**: Image name for Azure disks.

• **Azure Disk Container Name**: Container name for Azure disks.

• **Azure Disk Account Prefix**: Prefix for randomly-generated Azure disk account names.

• **Node Selection**: Type of node selection used by **Auto Scaler**. The values this can take are:
  - **Alphabetically**: This means that **Auto Scaler** picks next node to start according to the node name.
  - **Randomly**: This means that the nodes are picked randomly, which helps make the node usage more even.
  - **Uptime**: This means that the nodes that have been used the least amount of time are picked first.

• **Node Selection Uptime Period**: If **Node Selection** is set to **uptime**, then the **Node Selection Uptime Period** is the time period from now into the past, over which **Auto Scaler** calculates the total uptime for nodes. So, with the default setting of 2w this means that the uptime of nodes is calculated over the last 2 weeks.

An overview of the parameters and submodes is given next. An example showing how they can be configured is given afterwards, in section 8.3.

**ScaleServer Role Global Parameters**

The **ScaleServer** role has the following global parameters for controlling the **cm-scale** service itself:

• **Debug**: Print debug messages to the log.

• **Dry Run**: If set, then the service runs in dry run mode. In this mode it may claim that actions have been carried out on the nodes that use the **cm-scale** service, however, no action is actually carried out on nodes. This mode is useful for demonstration or debug purposes

• **Run Interval**: interval, in seconds, between **cm-scale** decision-making

**ScaleServer Role Submodes**

Within the **ScaleServer** role are the following three submodes:

• **advancedsettings**: allows some advanced properties to be set for **cm-scale**, using the parameters displayed on page 381.

• **resourceproviders**: defines the nodes used by **cm-scale**. More explicitly, this submode is used to define resource provider objects. The resource providers can be added as **static** or **dynamic** types, and can then have nodes and settings defined within them. The nodes allocated to these resource provider objects are what provide resources to **cm-scale** when that resource provider is requested.
8.2 Configuration

- **engines**: define the engines used by cm-scale. This can be an instance of the type hpc, generic, or kubernetes (page 388).
  - trackers (within engines submode): define the trackers used by cm-scale (page 390)

The parameters are enforced only when the next decision-making iteration takes place.

8.2.2 Resource Providers

The cm-scale service allows nodes to change state according to the workload demand. These managed nodes are defined by the administrator within the resourceproviders submode of ScaleServer. NVIDIA Bright Cluster Manager 9.2 supports two types of resource providers: static and dynamic node providers.

**Static Node Provider**

When managed nodes are well-known and will not be extended or shrunk dynamically, then a static node provider can be used. Specifying settings for the static node provider allows cm-scale to power on, power off, or re-purpose nodes, based on nodegroups or a list of nodes nodes specified with a node list syntax (page 44).

The static node provider supports the following properties:

- **Enabled**: The static node provider is currently enabled.
- **Nodes**: A list of nodes managed by cm-scale. These can be regular local compute nodes (nodes) or cluster extension cloud compute nodes (cnodes). For the purposes of this section on cm-scale, these compute nodes can conveniently be called nodes and cnodes. Since compute nodes are typically the most common cluster nodes, significant resources can typically be saved by having the cm-scale service decide on whether to bring them up or down according to demand.
  - cnodes can be cloned and terminated as needed. Cloning and terminating saves on cloud storage costs associated with keeping virtual machine images.
  - regular local compute nodes can be started and stopped as needed. This reduces power consumption.
- **Nodegroups**: List of nodegroups with nodes to be managed by cm-scale. Node groups are classed into types. The class of node group types is independent of the class of node types, and should not be confused with it. Node types can be physicalnode, cloudnode, gpuunit, genericdevice, headnode, ethernetswitch, ibswitch and others, and are shown in the first column of the output of the default list command in device mode. Node group types currently (May 2017) consist only of Normal and Storage pair. Only node groups of type Normal can currently be managed by cm-scale.
- **Priority**: The provider priority. Nodes in the pool of a provider with a higher priority are used first by workloads. By default a resource provider has a priority value 0. These priority values should not be confused with the fairsharing priorities of page 388.

**Dynamic Node Provider**

When managed nodes can be cloned or removed from the configuration, then a dynamic node provider should be used. A compute node that is managed by cm-scale as a dynamic node provider is configured as a template node within the dynamic submode of the ScaleServer role.

The dynamic node provider supports the following properties:

- **Template Node**: A node that will be used as a template for cloning other nodes in the pool. The following restrictions apply to the template node:
  - A workload manager client role must be assigned with a positive number of slots.
- New node names should not conflict with the node names of nodes in a nodegroup defined for the queue.
- A specific template node is restricted to a specific queue.

A template node only has to exist as an object in the cluster manager, with an associated node image. A template node does not need to be up and running physically in order for it to be used to create clones. Sometimes, however, an administrator may want it to run too, like the other nodes that are based upon it, in which case the Start Template Node and Stop Template Node values apply:

- **Start Template Node**: The template node specified in the Template Node parameter is also started automatically on demand.
- **Stop Template Node**: The template node specified in the Template Node parameter is also stopped automatically on demand.
- **Never Terminate**: Number of cloud nodes that are never terminated even if no jobs need them. If there are this number or fewer cloud nodes, then cm-scale no longer terminates them. Cloud nodes that cannot be terminated can, however, still be powered off, allowing them to remain configured in the cluster manager. As an aside, local nodes that are under cm-scale control are powered off automatically when no jobs need them, regardless of the Never Terminate value.
- **Never Terminate Nodes**: A list of nodes specified with a node list syntax (page 44). These cloud nodes are never terminated, even if no jobs need them. Cloud nodes that cannot be terminated can, however, still be powered off. The nodes must already exist in the cluster manager configuration when Never Terminate Nodes is configured.
- **Enabled**: Node provider is currently enabled.
- **Priority**: Node provider priority.
- **Node Range**: Range of nodes that can be created and managed by cm-scale.
- **Network Interface**: Which node network interface is changed on cloning (incremented).
- **Remove Nodes**: Should the new node be removed from the cluster manager when the node terminates? If the node is not going to be terminated, but just stopped, then it is never removed.
- **Leave Failed Nodes**: If nodes are discovered to be in a state of INSTALLER_FAILED or INSTALLER_UNREACHABLE (section 5.5.4) then this setting decides if they can be left alone, so that the administrator can decide what do with them later on.
- **Default Resources**: List of default resources, in format `[name=value]`.
  - `cpu`: value is the number of CPUs
  - `mem`: value is in bytes

These must be set when no real node instance is associated with a node defined in the cluster manager.

**Extra Nodes Settings For Node Providers**

Both the dynamic and static node providers support extra node settings. If configured, then cm-scale can start the extra nodes before the first workload is started, and can stop them after the last job from the managed queue is finished.

The most common use case scenario for extra nodes in the case of cloud nodes is a cloud director node. The cloud director node provisions cloud compute nodes and performs other management operations in a cloud.
In the case of non-cloud non-head nodes, extra nodes can be, for example, a license server, a provisioning node, or an additional storage node.

The configuration settings include:

- **Extra Nodes**: A list of extra nodes.
- **Extra Node Idle Time**: The maximum time, in seconds, that extra nodes can remain unused. The \texttt{cm-scale} service checks for the existence of queued and active workloads using the extra node, when the time elapsed since the last check reaches \texttt{Extra Node Idle Time}. If there are workloads using the extra node, then the time elapsed is reset to zero and a time stamp is written into the file \texttt{cm-scale.state} under the directory set by the \texttt{Spool} role parameter. The time stamp is used to decide when the next check is to take place. Setting \texttt{Extra Node Idle Time=0} means the extra node is stopped whenever it is found to be idle, and started again whenever workloads require it, which may result in a lot of stops and starts.
- **Extra Node Start**: Extra node is started by \texttt{cm-scale} before the first compute node is started.
- **Extra Node Stop**: Extra node is stopped by \texttt{cm-scale} after the last compute node stops.

**Additional Settings For Node Providers**

All of the node providers include the following settings that allows the Auto Scaler behavior to be tuned:

- **Keep Running**: Nodes that should not be stopped or terminated even if they are unused (range format).
- **Shutdown Before Power Off**: Shutdown nodes instead of just power off, and wait until a set timeout before doing a hard power off.
- **Shutdown Timeout**: Shutdown timeout before powering off.
- **Allocation Prolog**: Script that is executed when a node is allocated to a workload.
- **Allocation Epilog**: Script that is executed when a node is deallocated.
- **Long starting node action**: Action that is applied to a long starting node. A long starting node is a node that takes too long to start. Options:
  - \texttt{none} (default)
  - \texttt{power off}
  - \texttt{terminate} (applied to dynamic node provider only)
- **Long starting node timeout**: How long Auto Scaler should wait before the action is applied for a long starting node.

The following table summarizes the default attributes in \texttt{cmsh} for the resource providers, along the path \texttt{cmsh->device[]->roles->scaleserver->resourceproviders[dynamic/static]}:

<table>
<thead>
<tr>
<th>Parameter</th>
<th>\texttt{static}</th>
<th>\texttt{dynamic}</th>
</tr>
</thead>
<tbody>
<tr>
<td>Revision</td>
<td>\texttt{static}</td>
<td>\texttt{dynamic}</td>
</tr>
<tr>
<td>Type</td>
<td>\texttt{static}</td>
<td>\texttt{dynamic}</td>
</tr>
<tr>
<td>Enabled</td>
<td>yes</td>
<td>yes</td>
</tr>
<tr>
<td>Priority</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>Whole Time</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>Stopping Allowance Period</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>Keep Running</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Extra Node</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
In the preceding table, the entry N/A means that the parameter is not available for the corresponding resource provider.

### 8.2.3 Time Quanta Optimization

Time quanta optimization is an additional feature that cm-scale can use for further cost-saving with certain cloud providers.

For instance, Amazon charges per whole unit of time, or time quantum, used per cloud node, even if only a fraction of that unit of time was actually used. The aim of the cluster manager’s time quanta optimization is to keep a node up as long as possible within the already-paid-for time quantum, but without incurring further cloud provider charges for a node that is not currently useful. That is, the aim is to:

- keep a node up if it is running jobs in the cloud
- keep a node up if it is not running jobs in the cloud, if its cloud time has already been paid for, until that cloud time is about to run out
- take a node down if it is not running jobs in the cloud, if its cloud time is about to run out, in order to avoid being charged another unit of cloud time

Time quanta optimization is implemented with some guidance from the administrator for its associated parameters. The following parameters are common for both static and dynamic node resource providers:

- **Whole Time.** A compute node running time (in minutes) before it is stopped if no workload requires it. Currently (May 2017) Amazon’s time quantum is 60 minutes. By default, the cluster manager uses a value of **Whole Time=0**, which is a special value that means **Whole Time** is ignored. Ignoring it means that the cluster manager does no time quanta optimization to try to optimize how costs are minimized, but instead simply takes down nodes when they are no longer running jobs.

- **Stopping Allowance Period.** A time (in minutes) just before the end of the **Whole Time** period, prior to which all power off (or terminate) operations must be started. The parameter associated
with time quanta optimization is the Stopping Allowance Period. This parameter can also be set by the administrator. The Stopping Allowance Period can be understood by considering the last call time period. The last call time period is the period between the last call time, and the time that the next whole-time period starts. If the node is to be stopped before the next whole-time charge is applied, then the last call time period must be at least more than the maximum time period that the node takes to stop. The node stopping period in a cluster involves cleanly stopping many processes, rather than just terminating the node instance, and can therefore take some minutes. The maximum period in minutes allowed for stopping the node can be set by the administrator in the parameter Stopping Allowance Period. By default, Stopping Allowance Period=0. Thus, for nodes that are idling and have no jobs scheduled for them, only if the last call time period is more than Stopping Allowance Period, does cm-scale stop the node.

The preceding parameters are explained next. Figure 8.20 illustrates a time line with the parameters used in time quanta optimization.

The algorithm that cm-scale follows, with and without time quanta optimization, can now be described using the two parameters explained so far:

1. cm-scale as part of its normal working, checks every Run Interval seconds to see if it should start up nodes on demand or shut down idling nodes.
2. If it sees idling nodes, then:
   (a) If Whole Time has not been set, or is 0, then there is no time quanta optimization that takes place. The cm-scale service then just goes ahead as part of its normal working, and shuts down nodes that have nothing running on them or nothing about to run on them.
   (b) If a non-zero Whole Time has been set, then a time quanta optimization attempt is made. The cm-scale service calculates the time period until the next time quantum from public cloud starts. This time period is the current closing time period. Its value changes each time that cm-scale is run. If
      • the current closing time period is long enough to let the node stop cleanly before the next time quantum starts, and
      • the next closing time period—as calculated by the next cm-scale run but also running within the current time quantum—is not long enough for the node to stop cleanly before the next quantum starts
then the current closing time period starts at a time called the last call.

In drinking bars, the last call time by a bartender allows some time for a drinker to place the final orders. This allows a drinker to finish drinking in a civilized manner. The drinker is meant to stop drinking before closing time. If the drinker is still drinking beyond that time, then a vigilant law enforcement officer will fine the bartender.

Similarly, the last call time in a scaling cluster allows some time for a node to place its orders to stop running. It allows the node to finish running cleanly. The node is meant to stop running before the next time quantum starts. If the node is still running beyond that time, then a vigilant cloud provider will charge for the next whole time period.

The last call time is the last time that \texttt{cm-scale} can run during the current whole-time period and still have the node stop cleanly within that current whole-time period, and before the next whole-time period starts. Thus, when \texttt{Whole Time} has been set to a non-zero time:

i. If the node is at the last call time, then the node begins with stopping

ii. If the node is not at the last call time, then the node does not begin with stopping

The algorithm goes back again to step 1.

8.2.4 Fairsharing Priority Calculation And Node Management

At intervals of \texttt{Run Interval}, \texttt{cm-scale} collects workloads using trackers configured in the \texttt{ScaleServer} role, and puts all the workloads in a single internal queue. This queue is then sorted by priorities. The priorities are calculated for each workload using the following fairsharing formula:

\[ p_{ij} = k_1 \times a_i + k_2 \times b_j + k_3 \times c_j \]  
\hspace{1cm} (8.1)

where:

- \( p_{ij} \) is the global priority for the \( i \)-th workload of the \( j \)-th engine. Its value is used to re-order the queue.
- \( k_1 \) is the age factor. This is the \texttt{agefactor} parameter that can be set via \texttt{cmsh} in the \texttt{engine} submode of the \texttt{ScaleServer} role. Usually it has the value 1.
- \( a_i \) is the age of the workload. That is, how long has passed since the \( i \)-th job submission, in seconds. This typically dominates the priority calculation, and makes older workloads a higher priority.
- \( k_2 \) is the external priority factor. It is a floating point number in the range \([0, 1]\), and is the \texttt{External Priority Factor} parameter that can be set via \texttt{cmsh} in the \texttt{engine} submode of the \texttt{ScaleServer} role.
- \( b_j \) is the workload priority retrieved from the engine.
- \( k_3 \) is the engine factor. It is a floating point number in the range \([0, 1]\), and is the \texttt{enginefactor} parameter in the \texttt{engine} submode of the \texttt{ScaleServer} role.
- \( c_j \) is the engine priority. This is the \texttt{priority} parameter in the \texttt{engine} submode of the \texttt{ScaleServer} role.

When all the workload priorities are calculated and the queue is re-ordered, then \texttt{cm-scale} starts to find appropriate nodes for workloads. The workloads are selected in order, from the top of the queue where the higher priority workloads are, to the bottom. This way a higher priority engine has a greater chance of getting nodes for its workloads than a lower priority engine.

The factors \( k_1, k_2 \) and \( k_3 \) in the equation 8.1 allow the significance of the related priority value in the final result to be controlled. For example if only the priority fetched from the engine should be taken into account, then \( k_1 \) and \( k_3 \) should be set to 0, and \( k_2 \) to 1. Or, for example, when both the age and engine priorities should be treated as equally important, then \( k_1 \) and \( k_3 \) can be set to 0.5 and \( k_2 \) to 0.

8.2.5 Engines

Each workload engine considered by \texttt{cm-scale} must be configured within the \texttt{engines} submode within the \texttt{ScaleServer} role. NVIDIA Bright Cluster Manager 9.2 supports the following workload engines:
8.2 Configuration

- Slurm
- PBS (OpenPBS and PBS Professional)
- LSF
- UGE

Engines can be of three types:

- **hpc**: for all HPC (High Performance Computing) workload managers
- **kubernetes**: for Kubernetes
- **generic**: for a generic type

**Common Parameters For The cm-scale Engines**

All three engine types have the following parameters and submode in common, although their values may differ:

- **Workloads Per Node**: The number of workloads that can be scheduled to run on the same node at the same time.
  - For a Kubernetes engine this parameter restricts only the number of either the Kubernetes controllers, or the individual jobs (without any controller) per node. It does not restrict the total number of pods that can be mapped per node by the cm-scale scheduler.
  - For example, a Kubernetes job, or Job with a capital 'J' in Kubernetes terminology, may consist of many pods. Then, if Workload Per Node is, for example, 2, then only 2 Jobs can be mapped to the node.
  - The number of pods is also taken into account by cm-scale, but this number is taken from the Kubernetes::Node role, where the Max Pods option can be set. If the role is not assigned to a node, then cm-scale assumes that there is no possibility for any pods to run on the node.

- **Priority**: The engine priority
- **Age Factor**: Fairsharing coefficient for workload age
- **Engine Factor**: Fairsharing coefficient for engine priority
- **External Priority Factor**: Fairsharing coefficient for external priority significance
- **Max Nodes**: The maximum number of running nodes allowed
- **Trackers**: Enters the workload trackers submode

**Non-common parameters for the cm-scale engines**:

- **For the hpc engine:**
  - **WLM Cluster**: A workload manager cluster name. The name is set during workload manager setup as the instance name. In cmsh the WLM cluster names are listed under wlm mode. In Bright View they can be seen along the clickpath HPC→Wlm Clusters.

- **For the kubernetes engine**, the following parameters can be set:
  - **Cluster**: These are the Kubernetes clusters for which pods are to be tracked. The cluster manager allows multiple Kubernetes clusters to run on a single compute cluster. Kubernetes must be already set up before this setting is configured.
  - **CPU Busy Threshold**: The CPU load is a value that can range from 0 to 1. The CPU Busy Threshold value defines if the node is too busy for new pods. Its default value is: 0.9.
Memory Busy Threshold: The Memory load is a value that can range from 0 to 1. The Memory Busy Threshold defines if the node is too busy for new pods. Its default value is: 0.9.

The CPU and Memory thresholds configured in the Kubernetes engine help cm-scale to decide when more nodes are needed. But cm-scale also retrieves the number of pods that are already running on the node and compares it with the Max Pods parameter that is configured in the Kubernetes::Node role assigned to the node. If the number of running pods is already equal or greater than the value of Max Pods, then (from the cm-scale point of view) the node cannot fit more pods, which means that a new node is needed.

8.2.6 Trackers
A workload tracker is a way to specify the workload and its node requirements to cm-scale. For HPC, the tracker may be associated with a specific queue, and cm-scale then tracks the jobs in that queue.

One or more trackers can be named and enabled within the trackers submode, which is located within the engines submode of the ScaleServer role. A queue (for workload managers) can be assigned to each tracker.

Example
There are three types of tracker objects supported in NVIDIA Bright Cluster Manager 9.2:

- **queue**: Used with an HPC type engine, where each workload (job) is associated with a particular queue. The attribute Type takes the value ScaleHpcQueueTracker, and the attribute Queue is set to the queue name for the job.

- **namespace**: Used with a kubernetes type engine.

- **generic**: Used with a generic type engine.

The following settings are common for both types of trackers:

- **Enabled**: Enabled means that workloads from this tracker are considered by cm-scale.

- **Allowed Resource Providers**: Only the specified resource providers (in the scaleserver role) will be used for a workload of this tracker (if empty than all allowed).

- **Assign Category**: A node category name that should be assigned to the managed nodes. When a node is supposed to be used by a workload, then cm-scale should assign the node category to that node. If the node is already running, and has no workloads running on it, but its category differs from the category specified for the jobs of the queue, then the node is drained, stopped and restarted on the next decision-making iteration of cm-scale, and takes on the assigned category. Further details on this are given in the section on dynamic nodes re-purposing, page 405.

- **Primary Overlays**: A list of configuration overlays.

  If a workload is associated with the tracker for which the overlays are specified by Primary Overlays, then the cluster manager-managed nodes are appended to those configuration overlays by cm-scale. This takes place after the node is removed from the previous overlays that it is associated with.

  If the node is already running, but has not yet been appended to the overlays specified for the workloads of the tracker, then the node is restarted when no other workloads run on the node, before going on to run the workloads with the new overlays.

- When a workload is associated with the tracker that has Primary Overlays set, then the pool of cm-scale-managed nodes is checked.

  The check is to decide on if a node is to be made available for the workload.
If the node is appended to the Primary Overlays already and is not running workloads, then cm-scale simply hands over a workload from the tracker to run on the node.

If the node is not appended to the Primary Overlays already, and is not running workloads, then cm-scale prepares and boots the node as follows:

* the node is drained and rebooted if it is up, or
* the node is undrained and merely booted if it is not up

The node is removed from any previous overlays that it was with, before booting up, and it is appended to the new overlays of Primary Overlays.

- The threshold settings:
  - Queue Length Threshold: number of pending workloads that triggers cloud bursting.
  - Age Threshold: workload pending time threshold, in seconds, that triggers cloud bursting for this workload.

The queue length and age thresholds allow the administrator to set when cm-scale starts or creates cloud bursting nodes. Both thresholds can be used at the same time, or just one of them can be used and the other can be ignored by setting it to 0.

If the queue length threshold is set, then cm-scale ignores pending workloads that are located higher (added later) than the threshold in the managed queue.

**Example**

Assuming there are 5 jobs in the queue, with job IDs 1, 2, 3, 4, and 5, where the 1st one is the first in the queue. If the queue length threshold is 3, then only jobs 1, 2 and 3 are taken into account, while jobs 4 and 5 are ignored.

**Example**

If the age threshold is set to 100, then only workloads older than 100 seconds are taken into account, while younger jobs are ignored.

The queue type tracker has only one parameter specific to the tracker: Queue. This is set to the workload queue that is being tracked.

**Namespace Tracker**

The namespace tracker of cm-scale is used to track Kubernetes workloads. It tracks Kubernetes jobs (via its Job controllers) and tracks individual pods. It does not start new nodes for pending pods owned by other types of Kubernetes pod controllers, such as ReplicaSet, DaemonSet, and so on. If non-Job controllers are running, then cm-scale will not stop or terminate those nodes.

The tracker settings in cmsh or Bright View include additional parameters that are in common with other trackers:

1. Controller Namespace: Tracks the Kubernetes namespace name. Only Kubernetes workloads from this namespace are tracked. To track more than one namespace, one tracker must be created per namespace.

2. Object: Type of Kubernetes objects to track. The cluster manager supports the following object types:

   (a) Job: A Kubernetes Job controller type represents one or several pods that are expected to eventually terminate. The controller nature makes this type of Kubernetes workload very suited to dynamic data centers.
(b) Pod: Individual pod, without any controller.

If the specified namespace does not exist in Kubernetes, then the tracked jobs or individual pods in this namespace are ignored by cm-scale.

**Generic Engine And Tracker**

The cm-scale service is able to deal with workloads that use various workload types. In order to add support of a new type of workload, the administrator

- adds an engine of type generic
- adds one or more trackers of type generic to the ScaleServer role
- implements Tracker and Workload classes in the Python programming language

When cm-scale starts a new iteration, it re-reads the engines and trackers settings from the ScaleServer role, and searches for the appropriate modules in its directories. In the case of a custom tracker, the module is always loaded according to the tracker handler path. When the tracker module is loaded, cm-scale requests a list of workloads from each of the tracker modules. So, the aim of the tracker module is to collect and provide the workloads to cm-scale in the correct format.

The path to the tracker module should be specified in the handler parameter of the generic tracker entity using cmsh or Bright View as follows:

**Example**

```
[root@bright92 ~]# cmsh
[bright92] % device roles master
[bright92->device[bright92]->roles]% use scaleserver
[bright92->device[bright92]->roles[scaleserver]]% engines
...->roles[scaleserver]->engines% add generic myengine
...% ->roles*[scaleserver*]->engines*[myengine*]% trackers
...*[myengine*]->trackers% add generic mytracker
...*[mytracker*]% set handler /cm/local/apps/cm-scale/examples/custom_tracker/tracker.py
...% ->roles*[scaleserver*]->engines*[myengine*]->trackers*[mytracker*]% commit
...%->roles*[scaleserver*]->engines*[myengine*]->trackers*[mytracker*]%
```

In the preceding example, the handler file `/examples/tracker.py` is an example that is provided with the cm-scale package. Another example module file provided with the package is `/examples/workload.py`, which implements the `ExampleWorkload` class. Together, the two examples can be used to generate any amount of simple workloads during each cm-scale iteration. It is recommended to use the tracker and workload classes as templates for custom tracker and workload modules created by the administrator.

The generic engine does not have any specific parameters associated with its type. It only has parameters common to all of the engine types.

If the generic engine is configured with at least one generic tracker in its role, then cm-scale loads the handler module and uses two functions that are implemented in the class. The class name can be chosen arbitrarily, but should contain the string “Tracker”, without the quotes. The two class functions used are:

1. `__init__`: initializes the tracker object. This can be omitted if there are no additional data values to initialize.
2. `get_workloads`: returns a list of objects belonging to a new class inherited from the `Workload` class. This new class should be created by the administrator.

The new workload class must provide the following functions and properties. The class name can be chosen arbitrarily:

- `__init__`: initializes the workload object.
- `get_workloads`: returns a list of objects belonging to a new class inherited from the `Workload` class. This new class should be created by the administrator.

The new workload class must provide the following functions and properties. The class name can be chosen arbitrarily:

- `__init__`: initializes the workload object.
- `get_workloads`: returns a list of objects belonging to a new class inherited from the `Workload` class. This new class should be created by the administrator.
1. `__init__`: initializes the workload object.

2. `to_string`: returns a string that identifies the workload. This is printed to the log file.

3. `begin_timestamp`: property that returns a unix timestamp that should be >0 if the workload is not allowed to start before that time. If it is 0 then it is ignored by `cm-scale`.

For example, the following very simple tracker and workload classes can be implemented:

**Example**

class ExampleTracker(Tracker):
    def get_workloads(self):
        return [ExampleWorkload(self)]

class ExampleWorkload(Workload):
    def __init__(self, tracker):
        Workload.__init__(self, tracker)
        self.set_id("1")
        self._update_state()
        self._update_age()
        self._update_resources()

    def to_string(self):
        return "workload %s" % self._id

    def _update_state(self):
        self._set_pending()

    def _update_age(self):
        self._age = 0

    def _update_resources(self):
        node_res = NodeResource("*")
        cpus_res = CpusResource(1)
        node_res.add_resource(cpus_res)
        engine_res = EngineResource("myengine")
        node_res.add_resource(engine_res)

        self.add_resource(node_res)

The classes should be located in different files, as Python module files. It is recommended, but not required, to keep both the files in the same directory. The `ExampleWorkload` class initializes the workload object with a state, age, and required resources. These values are described next.

**State:** The state can be `pending`, `running` or `failed`, which can be set with these appropriate functions:

- `self._set_pending()`
- `self._set_running()`
- `self._set_failed()`
If the state is **running**, then the workload is treated as one that occupies the nodes defined in the resources list. Each NodeResource object in the resources thus represents one occupied node.

If the state is **pending**, then the workload is treated as one that waits for free nodes. In this case `cm-scale` tries to find (start or clone) some more nodes in order to allow the workload engine to start this workload.

The **failed** workload state is considered by `cm-scale` as exceptional. Such a workload is logged in the log file, but is not considered when `cm-scale` decides what nodes to start or clone. Any other workload state is also ignored by `cm-scale`.

**Age:** The `age` defines how many seconds the workload is waiting for its resources since being added to the engine. Usually the engine can provide such information, so in the example `age` is set to 0, which means the workload has been added to the workload engine just now. The `age` is used in the fairsharing workload priority calculation (page 388). The value of `age` is not re-calculated by `cm-scale` after a while. This means that the number that the module sets in the class is used during the iteration, and then forgotten by `cm-scale` until the workload object is recreated from scratch on the next iteration.

**Resources:** The Workload class (a base class) includes the `resources` property with list types. This list includes resource objects that are used by `cm-scale` in order to find appropriate nodes for the workload. The top level resource type is always NodeResource. There can be one or several node resources requested by the workload.

If the node names are known, then one NodeResource object is created per compute node.

Otherwise a single NodeResource object is used as many times as the number of requested nodes, with the name set to `*`, which is treated by `cm-scale` as any suitable node. The number of nodes can be set in the NodeResource object with the `set_amount(number)` function of the resource.

In the preceding example one (any) node resource is added to the workload request, and the requirement for CPU (cores) number is set to 1. The engine resource is used in order to restrict the mapping of workloads from different engines to one node. Thus if a node has this resource assigned, then the node can take on the workload. If no engine resource is assigned to the node, then it can also take on the workload, but the engine resource of the workload is assigned to the node before other workloads are considered.

The resource types that can be added to the workload are defined in the Python module `core/resource.py`:

- **NodeResource**: top level resource, contains all other resources.
- **CpusResource**: defines the number of cpu cores required or already used by the workload.
- **CategoryResource**: node category required by the workload.
- **OverlayResource**: required configuration overlay.
- **QueueResource**: HPC queue that the workload (job) belongs to. Used only with engines that support queues.
- **EngineResource**: engine name that the workload belongs to.
- **FeatureResource**: required node feature (node property, in other terminology) that should be supported by the engine.

Custom resource types are not supported for now.

In order to drain a node in the custom engine before the node is stopped, and to undrain it before the node is started, the administrator can write and configure three scripts:

- **Drain script**: called before a node is drained by Auto Scaler.
• **Undrain script**: called before node is undrained by Auto Scaler.

• **Drain status script**: called when Auto Scaler retrieves information about the current node drain status.

Either all of the three scripts must be configured, or none of them.

It is useful to drain and undrain the nodes in order to ensure that the engine does not start new jobs in time period between the instant that Auto Scaler decides to stop the node, and the instant that the actual power operation is performed.

The scripts are configured in the file:

```
/cm/local/apps/cm-scale/lib/python3.9/site-packages/cmscale/config.py
```

with the `GENERIC_DRAIN_COMMANDS` parameter appended to the `opts` dictionary:

**Example**

```
"GENERIC_DRAIN_COMMANDS": {
    "MyEngine": {
        "drain": "/cm/local/apps/cm-scale/examples/custom_drain/drain.py",
        "undrain": "/cm/local/apps/cm-scale/examples/custom_drain/undrain.py",
        "status": "/cm/local/apps/cm-scale/examples/custom_drain/drainstatus.py"
    }
},
```

Here, for each generic engine, a new dictionary is created that includes three items that correspond to, and specify, the script paths. In the preceding example `MyEngine` is the engine name, and should be the same as that defined in the `ScaleServer` role. If more than one generic engine is used then all of them can be added to `GENERIC_DRAIN_COMMANDS`.

It should be noted that if `GENERIC_DRAIN_COMMANDS` is defined in `config.py`, then CMDaemon does not drain, or undrain, via `cm-scale`

All three scripts accept the same set of parameters, following the form:

```
<script name> <engine name> <host name> [host name ... ]
```

**Example**

```
drain.py MyEngine node001 node002 node003
```

Each of those three scripts print the following information to standard output:

• **stdout**: JSON structure that represents a map: hostname -> latest (new) drain status. For example:

**Example**

```
{"node001": 2, "node002": 2, "node003": 2}
```

Here the numbers are enum values defined in `pythoncm` in the `DrainResult` class.

• **stderr**: debug logs that are appended to `cm-scale.log`. 

Enabling Node Shutdown

By default, `cm-scale` powers off nodes belonging to a resource pool once there is no more workload for them. Resource providers can also enable shutdown, to allow the node to terminate gracefully. Shutdown has two options that set its behavior directly:

- **Shutdown Enable**: If set to `yes`, then the shutdown command is run to terminate the system services first, and after that a command is run to power off the system. A waiting time of **Shutdown Timeout** seconds takes place between the two commands.

- **Shutdown Timeout**: The number of seconds to wait before powering off a node that is in a shutdown state.

It may take more than **Shutdown Timeout** seconds for a node to power off, depending on the **Run Interval** setting. For example, if **Shutdown Timeout** is 60, and **Run Interval** 50, then effectively the **Shutdown Timeout** is 100, because the power off event only happens during an iteration execution of `cm-scale`.

8.3 Examples Of `cm-scale` Use

8.3.1 Simple Static Node Provider Usage Example

The example session that follows explains how a static node provider (page 383) can be configured and used with `cm-scale`. The session considers a default cluster with a head node and 5 regular nodes which have been previously defined in the cluster manager configuration. 3 of the regular nodes are powered down at the start of the run. The power control for the nodes must be functioning properly, or otherwise `cm-scale` cannot power nodes on and off.

The head node has the Slurm server role by default, and the regular nodes run with the Slurm client role by default. So, on a freshly-installed cluster, the `roleoverview` command should show something like:

Example

```
[bright92->device[bright92]]% roleoverview | head -2 ; roleoverview | grep slurm
Role       Nodes       Categories Nodes up
----------  ----------  ------------  -------
slurmclient node001..node005 default 2 of 5
slurmserver bright92              1 of 1
```

A test user, `fred` can be created by the administrator (section 6.2), and an MPI `hello` executable based on the `hello.c` code (from section 3.5.1 of the User Manual) can be built:

Example

```
[fred@bright92 ~]$ module add shared openmpi/gcc/64/1.10.3 slurm
[fred@bright92 ~]$ mpicc hello.c -o hello
```

A batch file `slurmhello.sh` (from section 5.3.1 of the User Manual) can be set up. Restricting it to 1 process per node so that it spreads over nodes easier for the purposes of the test can be done with the settings:

Example

```
[fred@bright92 ~]$ cat slurmhello.sh
#!/bin/sh
#SBATCH -o my.stdout
#SBATCH --time=30 #time limit to batch job
#SBATCH --ntasks=1
```

8.3 Examples Of cm-scale Use

#SBATCH --ntasks-per-node=1
module add shared openmpi/gcc/64/1.10.1 slurm

mpirun /home/fred/hello

The user fred can now flood the default queue, defq, with the batch file:

Example

[fred@bright92 ~]$ while (true); do sbatch slurmhello.sh; done

After putting enough jobs into the queue (a few thousand should be enough, and keeping it less than 5000 would be sensible) the flooding can be stopped with a ctrl-c.

The activity in the queue can be watched:

Example

[root@bright92 ~]# watch "squeue | head -3 ; squeue | tail -3"

Every 2.0s: squeue | head -3 ; squeue | tail -3

The preceding indicates that node001 and node002 are being kept busy running the batch jobs, while the remaining nodes are not in use.

The administrator can check on the job status via the job metrics of cmsh too, using the options to the filter command, such as --pending or --running:

Example

[root@bright92 ~]# cmsh -c "wlm use slurm; jobs; watch filter --running -u fred"

Every 2.0s: filter --running --u fred

Job ID Job name User Queue Submit time Start time End time Nodes Exit code

and eventually, when jobs are no longer running, it should show something like:

[root@bright92 ~]# cmsh -c "wlm use slurm; jobs; watch filter --running -u fred"

Every 2.0s: filter --running

No jobs found

So far, the cluster is queuing or running jobs without cm-scale being used.

The next steps are to modify the behavior by bringing in cm-scale. The administrator assigns the ScaleServer role to the head node. Within the role a new static node provider, Slurm engine, and queue tracker for the defq are set as follows:

Example
The nodes node001..node005 should already be in the queue defq, as assigned to them by default when they were assigned the SlurmClient role. With these settings, they can now be powered up or down on demand by cm-scale service, depending on the number of jobs that are pending. When the new ScaleServer role is committed in cmsh or Bright View, then the cm-scale service is started. If needed, the administrator can check the log file /var/log/cm-scale to see what the service is doing.

On each iteration cm-scale checks whether the node states should be changed. Thus after a while, the nodes node003..node005 are started. Once up, they can start to process the jobs in the queue too.

Watching the running jobs should show something like:

Example

```
[root@bright92 ~]# cmsh -c "wlm use slurm; jobs ; watch filter --running"
```

<table>
<thead>
<tr>
<th>Job ID</th>
<th>Job name</th>
<th>User</th>
<th>Queue</th>
<th>Submit time</th>
<th>Start time</th>
<th>Nodes</th>
<th>Exit code</th>
</tr>
</thead>
<tbody>
<tr>
<td>16498</td>
<td>slurmhello.sh</td>
<td>fred</td>
<td>defq</td>
<td>07/05/2017 12:44:20</td>
<td>07/05/2017 13:20:58</td>
<td>node003</td>
<td>0</td>
</tr>
<tr>
<td>16499</td>
<td>slurmhello.sh</td>
<td>fred</td>
<td>defq</td>
<td>07/05/2017 12:44:20</td>
<td>07/05/2017 13:20:58</td>
<td>node002</td>
<td>0</td>
</tr>
</tbody>
</table>

Eventually, cm-scale finds that all jobs have been dealt with, and the nodes are then powered down.

**High-availability And Using A Configuration Overlay For The ScaleServer Role**

For high-availability clusters, where there are two head nodes, the scaleserver should run on the active head node. One labor-intensive way to set this up is to assign the service to both the head nodes, and match the scaleserver settings on both head nodes. A simpler way is to define a configuration overlay for the head nodes for the scaleserver. If the head nodes are bright92-1 and bright92-2, then a configuration overlay called brightheads can be created and assigned the service as follows:

Example

```
[bright92-1]% configurationoverlay add brightheads
[bright92-1->configurationoverlay*[brightheads*]]% append nodes bright92-1 bright92-2
[bright92-1->configurationoverlay*[brightheads*]]% roles
[bright92-1->configurationoverlay*[brightheads*]->roles% assign scaleserver
[bright92-1->configurationoverlay*[brightheads*]->roles*[scaleserver*]]%
```

The scaleserver can then be configured within the configuration overlay instead of on a single head as was done previously in the example of page 397. After carrying out a commit, the scaleserver settings modifications are then mirrored automatically between the two head nodes.
Outside the scaleserver settings, one extra modification is to set the cm-scale service to run on a head node if the head node is active. This can be done with:

Example

[bright92-1->configurationoverlay[brightheads]-->roles[scaleserver]]% device services bright92-1
[bright92-1->device[bright92-1]-->services]% use cm-scale
[bright92-1->device[bright92-1]-->services[cm-scale]]% set runif active
[bright92-1->device[bright92-1]-->services[cm-scale]]% commit
[bright92-1->device[bright92-1]-->services]% use cm-scale
[bright92-1->device[bright92-1]-->services[cm-scale]]% set runif active
[bright92-1->device[bright92-1]-->services[cm-scale]]% commit

The result is a scaleserver that runs when the head node is active.

8.3.2 Simple Dynamic Node Provider Usage Example

The following example session explains how a dynamic node provider (page 383) can be configured and used with cm-scale. The session considers a default cluster with a head node and 2 regular nodes which have been previously defined in the cluster manager configuration, and also 1 cloud director node and 2 cloud compute nodes. The cloud nodes can be configured using cm-cluster-extension. Only the head node is running at the start of the session, while the regular nodes and cloud nodes are all powered down at the start of the run.

At the start, the device status shows something like:

Example

[bright92->device]% ds
eu-west-1-cnode001 ....... [ DOWN ] (Unassigned)
eu-west-1-cnode002 ....... [ DOWN ] (Unassigned)
eu-west-1-cnode003 ....... [ DOWN ] (Unassigned)
eu-west-1-director ....... [ DOWN ]
node001 .................. [ DOWN ]
node002 .................. [ DOWN ]
bright92 .................. [ UP ]

The power control for the regular nodes must be functioning properly, or otherwise cm-scale cannot power them on and off.

If the head node has the slurmserver role, and the regular nodes have the slurmclient role, then...
the roleoverview command should show something like:

Example

```
[bright92->device[bright92]]% roleoverview
Role     Nodes Categories                        Nodes up
----------------- -------------------------------------- ---------------------------- --------
boot         bright92                           1 of 1
cgroupsupervisor eu-west-1-cnode001..eu-west-1-cnode002 aws-cloud-director,default 1 of 6
             .eu-west-1-director.node001..node002 .eu-west-1-cloud-node .bright92
clouddirector eu-west-1-director               0 of 1
cloudgateway bright92                          1 of 1
login        bright92                          1 of 1
master       bright92                          1 of 1
monitoring   bright92                          1 of 1
provisioning eu-west-1-director,bright92       1 of 2
slurmclient  eu-west-1-cnode001..eu-west-1-cnode002 default.eu-west-1-cloud-node 0 of 3
             .node001..node002
slurmserver  bright92                          1 of 1
storage      eu-west-1-director,bright92       1 of 2
             aws-cloud-director
```

A test user, fred can be created by the administrator (section 6.2), and an MPI hello executable based on the hello.c code (from section 3.5.1 of the User Manual) can be built:

Example

```
[fred@bright92 ~]$ module add shared openmpi/gcc/64/1.10.3 slurm
[fred@bright92 ~]$ mpicc hello.c -o hello
```

A batch file slurmhello.sh (from section 5.3.1 of the User Manual) can be set up. Restricting it to 1 process per node so that it spreads over nodes easier for the purposes of the test can be done with the settings:

Example

```
[fred@bright92 ~]$ cat slurmhello.sh
#!/bin/sh
#SBATCH -o my.stdout
#SBATCH --time=30 #time limit to batch job
#SBATCH --ntasks=1
#SBATCH --ntasks-per-node=1
module add shared openmpi/gcc/64/1.10.1 slurm

mpirun /home/fred/hello
```

A default cluster can queue or run jobs without cm-scale being used. The default behavior is modified in the next steps, which bring in the cm-scale service:

The administrator assigns the ScaleServer role to the head node.

Example

```
[root@bright92 ~]# cmsh
[bright92]% device roles master
[bright92->device[bright92]->roles]% assign scaleserver
```

Within the assigned scaleserver role, a new dynamic node provider can be set, and properties for the dynamic pool of nodes can be set for the cloud compute nodes. Here the properties that are set are priority (page 383), templatenode (page 383), noderange (page 384), and extranodes (page 384).
8.3 Examples Of cm-scale Use

Example

[bright92->device*]->roles*[scaleserver*]->resources*% resourceproviders
...->roles*[scaleserver*]->resources*% add dynamic pool2
...resources*[pool2*]% set priority 2
...resources*[pool2*]% set noderange eu-west-1-cnode001..eu-west-1-cnode002
...resources*[pool2*]% set templateNode eu-west-1-cnode001
...resources*[pool2*]% set extranodes eu-west-1-director
...resources*[pool2*]% commit
...resources*[pool2*]%

The regular compute nodes, node001..node002 should be specified as nodes in the static pool.
The administrator may notice the similarity of dynamic and static pool configuration. The cluster
manager front end has deliberately been set up to present dynamic pool and static pool nodes to the
cluster administrator as two different configuration methods. This is because separating the pool types
as dynamic and static pools is simpler for the cluster administrator to deal with. This way, regular
compute nodes are treated, not as a special case of a dynamic pool, but simply as static pool nodes.
The fundamental reason behind this separate treatment is because physical nodes cannot “materialize”
dynamically with properties in the way the cloud compute nodes—which are virtualized nodes—can,
due to the need to associate a MAC address with a physical node.
Assigning regular compute nodes to a static pool can be done in a similar way to what was shown
before in the example on page 397.
Continuing with the current the session, the nodes node001..node002 are added to the static pool of
nodes, on-premises-nodes. For this example they are set to a lower priority than the cloud nodes:

Example

...->roles*[scaleserver*]->resources*% add static on-premises-nodes
...->roles*[scaleserver*]->resources*% set nodes node001..node002
...->roles*[scaleserver*]->resources*% set priority 1
...->roles*[scaleserver*]->resources*[on-premises-nodes*] % commit
...->roles*[scaleserver*]->resources*[on-premises-nodes*]%

What this lower priority means is that a node that is not up and is in the static pool of nodes, is only
powered on after all the cloud nodes are powered on and busy running jobs. If there happen to be nodes
from the static pool that are already up, but are not running jobs, then these nodes take a job, despite
the lower priority of the static pool, and irrespective of whether the dynamic pool nodes are in use.
Job priorities can be overridden in cm-scale by:

• allowing locations by setting Mix Locations to true (page 407) or
• pinning queues by setting Pin Queues to true (page 410)

A Slurm engine, and queue tracker for the defq are set as follows:

Example

...]->roles*[scaleserver*] % engines
...]->roles*[scaleserver*]->engines*[slurm2*] % add hpc slurm2
...]->roles*[scaleserver*]->engines*[slurm2*] % set wlmCluster slurm
...]->roles*[scaleserver*]->engines*[slurm2*] % trackers
...]->roles*[scaleserver*]->engines*[slurm2*]->trackers*% add queue tr2
...]->roles*[scaleserver*]->engines*[slurm2*]->trackers*[tr2*] % set queue defq
...]->roles*[scaleserver*]->engines*[slurm2*]->trackers*[tr2*] % commit
...]->roles*[scaleserver*]->engines*[slurm2*]->trackers*[tr2*]%
The nodes node001..node002 and eu-west-1-cnode001..eu-west-1-cnode002 should already be in the queue defq by default, ready to run the jobs:

Example

```
...->roles[scaleserver]->engines[slurm2]->trackers[tr2]]% wlm use slurm; jobqueue; get defq nodes eu-west-1-cnode001 eu-west-1-cnode002 node001 node002
```

The roleoverview (page 400) command is also handy for an overview, and to confirm that the role assignment of these nodes are all set to the SlurmClient role:

With these settings, the nodes in the dynamic pool can now be powered up or down on demand by cm-scale service, depending on the number of jobs that are pending. When the new ScaleServer role is committed in cmsh or Bright View, then the cm-scale is run periodically. Each time it is run, cm-scale checks whether the node states should be changed. If needed, the administrator can check the log file /var/log/cm-scale to see what the service is doing.

Job submission can now be carried out, and the scaleserver assignment carried out earlier scales the cluster to cope with jobs according to the configuration that has been carried out in the session.

Before submitting the batch jobs, the administrator or user can check the jobs that are queued and running with the squeue command. If there are no jobs yet submitted, the output is simply the squeue headers, with no job IDs listed:

Example

```
[fred@bright92 ~]$ squeue
JOBID PARTITION NAME USER ST TIME NODES NODELIST(REASON)
```

As in the previous example for the static pool only case (page 396), a way for user fred to flood the default queue defq is to run the batch file in a loop:

Example

```
[fred@bright92 ~]$ while (true); do sbatch slurmhello.sh; done
Submitted batch job 1
Submitted batch job 2
Submitted batch job 3
...
```

After putting enough jobs into the queue (a few thousand should be enough, not more than five thousand would be sensible), the flooding can be stopped with a ctrl-c.

The changes in the queue can be watched by user fred:

Example

```
[fred@bright92 ~]$ watch "squeue | head -5 ; squeue | tail -4"
Every 2.0s: squeue | head -5 ; squeue | tail -4 Wed Nov 22 16:08:52 2017
```

```
JOBID PARTITION NAME USER ST TIME NODES NODELIST(REASON)
 1  defq slurmhel fred PD 0:00 1 (Resources)
 2  defq slurmhel fred PD 0:00 1 (Resources)
 3  defq slurmhel fred PD 0:00 1 (Resources)
 4  defq slurmhel fred PD 0:00 1 (Resources)
3556 defq slurmhel fred PD 0:00 1 (Resources)
3557 defq slurmhel fred PD 0:00 1 (Resources)
3558 defq slurmhel fred PD 0:00 1 (Resources)
3559 defq slurmhel fred PD 0:00 1 (Resources)
```
The `head -4` and `tail -4` filters here are convenient for showing just the first 4 rows and last 4 rows of the very long `squeue` output, and skipping the bulk of the queue.

The preceding output illustrates how, with the jobs queued up, nothing is being processed yet from jobs number 1 to 3559 due to the resources not yet being available.

At this point `cm-scale` should have noticed that jobs are queued and that resources are needed to handle the jobs.

At the start of this example session the cloud director is not up. So, `cm-scale` powers it up. This can be seen by running the `ds` command, or from CMDaemon info messages:

```
[bright92->device] % ds | grep director
```

```
eu-west-1-director [ DOWN ]
then some time later:
    eu-west-1-director [ PENDING ] (External ip assigned: 34.249.166.63, setting up tunnel)
then some time later:
    eu-west-1-director [ INSTALLING ] (node installer started)
then some time later:
    eu-west-1-director [ INSTALLER_CALLINGINIT ] (switching to local root)
then some time later:
    eu-west-1-director [ UP ]
```

If the cloud director is yet to be provisioned to the cloud from the head node for the very first time ("from scratch"), then that can take a while. Then, because the cloud compute nodes are in turn provisioned from the cloud director, it takes a while for the cloud compute nodes to be ready to run the jobs. So, the jobs just have to wait around in the queue until the cloud compute nodes are ready, before they are handled. Fortunately, the startup of a cloud director is by default much faster after the very first time.

A quick aside about how provisioning is speeded up the next time around: The cloud compute nodes will be stopped if they are idle, and after there are no more jobs in the queue, because the jobs have all been dealt with. Then, when the `extranodeidletime` setting has been exceeded, the cloud director is also stopped. The next time that jobs are queued up, all the cloud nodes are provisioned from a stopped state, rather than from scratch, and so they are ready for job execution much faster. Therefore, unlike the first time, the jobs queued up the next time are processed with less waiting around.

Getting back to how things proceed in the example session after the cloud director is up: `cm-scale` then provisions the cloud compute nodes `eu-west-1-node001` and `eu-west-1-node002` from the cloud director.

**Example**

```
[bright92->device] % ds | grep cnode
```

```
eu-west-1-cnode001 ....... [ PENDING ] (Waiting for instance to start)
eu-west-1-cnode002 ....... [ PENDING ] (Waiting for instance to start)
then some time later:
    eu-west-1-cnode002 [ INSTALLING ] (node installer started)
eu-west-1-cnode001 [ INSTALLING ] (node installer started)
and so on
```

Once these cloud compute nodes reach the state of `UP`, they can start to process the jobs in the queue. The queue activity then would show something like:

**Example**

*when the dynamic pool nodes are being readied for job execution:*

```
[fred@bright92 ~]$ squeue | head -5 ; squeue | tail -4
```

```
JOBID PARTITION   NAME USER ST     TIME NODES Nodelist(REASON)
   1   defq   slurmhel  fred PD 0:00 1 (Resources)
```
then later:

<table>
<thead>
<tr>
<th>JOBID</th>
<th>PARTITION</th>
<th>NAME</th>
<th>USER</th>
<th>ST</th>
<th>TIME</th>
<th>NODES</th>
<th>NODELIST (REASON)</th>
</tr>
</thead>
<tbody>
<tr>
<td>11</td>
<td>defq</td>
<td>slurmhel</td>
<td>fred</td>
<td>CF</td>
<td>0:00</td>
<td>1</td>
<td>eu-west-1-cnode001</td>
</tr>
<tr>
<td>12</td>
<td>defq</td>
<td>slurmhel</td>
<td>fred</td>
<td>CF</td>
<td>0:00</td>
<td>1</td>
<td>eu-west-1-cnode002</td>
</tr>
<tr>
<td>13</td>
<td>defq</td>
<td>slurmhel</td>
<td>fred</td>
<td>CF</td>
<td>0:00</td>
<td>1</td>
<td></td>
</tr>
<tr>
<td>14</td>
<td>defq</td>
<td>slurmhel</td>
<td>fred</td>
<td>CG</td>
<td>0:00</td>
<td>1</td>
<td></td>
</tr>
<tr>
<td>3556</td>
<td>defq</td>
<td>slurmhel</td>
<td>fred</td>
<td>PD</td>
<td>0:00</td>
<td>1</td>
<td></td>
</tr>
<tr>
<td>3557</td>
<td>defq</td>
<td>slurmhel</td>
<td>fred</td>
<td>PD</td>
<td>0:00</td>
<td>1</td>
<td>(Priority)</td>
</tr>
<tr>
<td>3558</td>
<td>defq</td>
<td>slurmhel</td>
<td>fred</td>
<td>PD</td>
<td>0:00</td>
<td>1</td>
<td>(Priority)</td>
</tr>
<tr>
<td>3559</td>
<td>defq</td>
<td>slurmhel</td>
<td>fred</td>
<td>PD</td>
<td>0:00</td>
<td>1</td>
<td>(Priority)</td>
</tr>
</tbody>
</table>

then later, when `cm-scale` sees all of the dynamic pool is used up, the lower priority static pool gets started up:

<table>
<thead>
<tr>
<th>JOBID</th>
<th>PARTITION</th>
<th>NAME</th>
<th>USER</th>
<th>ST</th>
<th>TIME</th>
<th>NODES</th>
<th>NODELIST (REASON)</th>
</tr>
</thead>
<tbody>
<tr>
<td>165</td>
<td>defq</td>
<td>slurmhel</td>
<td>fred</td>
<td>CF</td>
<td>0:00</td>
<td>1</td>
<td>eu-west-1-cnode001</td>
</tr>
<tr>
<td>166</td>
<td>defq</td>
<td>slurmhel</td>
<td>fred</td>
<td>CF</td>
<td>0:00</td>
<td>1</td>
<td>node001</td>
</tr>
<tr>
<td>168</td>
<td>defq</td>
<td>slurmhel</td>
<td>fred</td>
<td>CG</td>
<td>0:00</td>
<td>1</td>
<td>node002</td>
</tr>
<tr>
<td>3556</td>
<td>defq</td>
<td>slurmhel</td>
<td>fred</td>
<td>PD</td>
<td>0:00</td>
<td>1</td>
<td>(Priority)</td>
</tr>
<tr>
<td>3557</td>
<td>defq</td>
<td>slurmhel</td>
<td>fred</td>
<td>PD</td>
<td>0:00</td>
<td>1</td>
<td>(Priority)</td>
</tr>
<tr>
<td>3558</td>
<td>defq</td>
<td>slurmhel</td>
<td>fred</td>
<td>PD</td>
<td>0:00</td>
<td>1</td>
<td>(Priority)</td>
</tr>
<tr>
<td>3559</td>
<td>defq</td>
<td>slurmhel</td>
<td>fred</td>
<td>PD</td>
<td>0:00</td>
<td>1</td>
<td>(Priority)</td>
</tr>
<tr>
<td>167</td>
<td>defq</td>
<td>slurmhel</td>
<td>fred</td>
<td>R</td>
<td>0:00</td>
<td>1</td>
<td>eu-west-1-cnode002</td>
</tr>
</tbody>
</table>

In `cmsh`, the priority can be checked with:

**Example**

```
[bright92 ->device[bright92] ->roles[scaleserver] ->resourceproviders] % list
Name (key) Priority Enabled
----------------- ------------ -------
on-premises-nodes 1 yes
pool2 2 yes
```

Also in `cmsh`, the jobs can be listed via the `jobs` submode:

**Example**

```
[bright92 ->wlm[slurm] ->jobs] % list | head -5 ; list | tail -4
Type Job ID User Queue Running time Status Nodes
----- ----- ----- ------ ------------ ---------- ------------------
Slurm 334 fred defq 1s COMPLETED eu-west-1-cnode001
Slurm 336 fred defq 1s COMPLETED node001
Slurm 3556 fred defq 0s PENDING
Slurm 3557 fred defq 0s PENDING
Slurm 3558 fred defq 0s PENDING
Slurm 3559 fred defq 0s PENDING
Slurm 335 fred defq 1s RUNNING eu-west-1-cnode002
```

[bright92 ->wlm[slurm] ->jobs] %
Eventually, when the queue has been fully processed, the jobs are all gone:

Example

```
[fred@bright92 ~]$ squeue
```

With the current configuration the cloud compute nodes in the dynamic pool pool2 are powered up before the regular compute nodes in the static pool on-premises-nodes. That is because the cloud compute nodes have been set by the administrator in this example to have a higher priority. This is typically sub-optimal, and is actually configured this way just for illustrative purposes. In a real production cluster, the priority of regular nodes is typically going to be set higher than that for cloud compute nodes, because using on-premises nodes is likely to be cheaper.

The administrator can also check on the job status via the job metrics of cmsh too, using the options to the filter command, such as --pending or --running:

Initially, before the jobs are being run, something like this will show up:

Example

```
[root@bright92 ~]# cmsh -c "wlm use slurm; jobs ; watch filter --running -u fred"
```

Then, eventually, when the jobs are being run, the cloud nodes, which have a higher priority, start job execution, so that the output looks like:

Example

```
Every 2.0s: filter --running -u fred Wed Nov 22 16:50:35 2017
No jobs found
```

and eventually the regular on-site nodes which are originally down are started up by the ScaleServer and are also listed.

8.4 Further cm-scale Configuration And Examples

8.4.1 Dynamic Nodes Re-purposing

Sometimes it is useful to share the same nodes among several queues, and reuse the nodes for jobs from other queues. This can be done by dynamically assigning node categories in cm-scale. Different settings, or a different software image, then run on the re-assigned node after re-provisioning.

The feature is enabled by setting Assign Category parameter in the tracker settings.

For example, the following case uses Slurm as the workload engine, and sets up two queues chem_q and phys_q. Assuming in this example that jobs that are to go to chem_q require chemistry software on the node, but jobs for phys_q require physics software on the node, and that for some reason the softwares cannot run on the node at the same time. Then, the nodes can be re-purposed dynamically. That is, the same node can be used for chemistry or physics jobs by setting up the appropriate configuration for it. In this case the same node can be used by jobs that require a different configuration, software, or even operating system. The trackers configuration may then look as follows:

Example
Parameter Value
-------------------------------- ------------------------------------------------
Type ScaleHpcQueueTracker
Name chem
Queue chem_q
Enabled yes
Assign Category chem_cat
Primary Overlays
Parameter Value
-------------------------------- ------------------------------------------------
Type ScaleHpcQueueTracker
Name chem
Queue chem_q
Enabled yes
Assign Category phys_cat
Primary Overlays

Assuming that initially there are two nodes, node001 and node002, both in category chem_cat. Then, when cm-scale finds a pending job in queue phys_q, it may decide to assign category phys_cat to either node001, or to node002. In this way the number of nodes serving queue phys_q increases and number of nodes serving chem_q decreases, in order to handle the current workload. When the job is finished, the old node category is not assigned back to the node, until a new job appears in chem_q and requires this node to have the old category.

### 8.4.2 Pending Reasons

This section is related only to HPC engines (workload managers). In this section, the term job is used instead of workload.

If cm-scale makes a decision on how many nodes should be started for a job, then it checks the status of the job first. If the job status is pending, then it checks the list of pending reasons for that job. The checks are to find pending reasons that prevent the job from starting when more free nodes become available.

A pending reason can be one of the following 3 types:

**Type 1**: allows a job to start when new free nodes become available

**Type 2**: prevents a job from starting on particular nodes only

**Type 3**: prevents a job from starting anywhere

Each pending reason has a text associated with it. The text is usually printed by the workload manager job statistics utilities. The list of pending reasons texts of types 1 and 2 can be found in the pending reasons exclude file, /cm/local/apps/cm-scale/lib/python3.9/site-packages/cmscale/trackers/hpc_queue/pending_reasons/WLM.exclude, where WLM is a name of workload manager specified in the configuration of the engine in ScaleServer role.

In the pending reasons exclude file, the pending reason texts are listed as one reason per line. The reasons are grouped in two sublists, with headers:

- [IGNORE_ALWAYS]
- [IGNORE_NO_NODE]
The \textit{\texttt{IGNORE\_ALWAYS}} sublist lists the type 1 pending reason texts. If a job has only this group of reasons, then \texttt{cm-scale} considers the job as ready to start, and attempts to create or boot compute nodes for it.

The \textit{\texttt{IGNORE\_NO\_NODE}} sublist lists the type 2 pending reason texts. If the reason does not specify the hostname of a new free node at the end of a pending reason after the colon (";"), then the job can start on the node. If the reason does specify the hostname of a new free node after the colon, and if the hostname is owned by one of the managed nodes—nodes that can be stopped/started/created by \texttt{cm-scale}—then the job is considered as one that is not to start, when nodes become available.

If a job has a pending reason text that is not in the pending reasons exclude file, then it is assumed to be a type 3 reason. New free nodes for such a job do not get the job started.

If there are several pending reason texts for a job, then \texttt{cm-scale} checks all the pending reasons one by one. If all reasons are from the \textit{\texttt{IGNORE\_ALWAYS}} or \textit{\texttt{IGNORE\_NO\_NODE}} sublists, and if a pending reason text matched in the \textit{\texttt{IGNORE\_NO\_NODE}} sublist does not include hostnames for the managed nodes, only then will the job be considered as one that can be started just with new nodes.

### Custom Pending Reasons

If the workload manager supports them, then custom pending reason texts are also supported. The administrator can add a pending reason text to one of the sections in the pending reasons exclude file.

The \texttt{cm-scale} service checks only if the pending reason text for the job starts with a text from the pending reasons file. It is therefore enough to specify just a part of the text of the reason in order to make \texttt{cm-scale} take it into account. Regular expressions are also supported. For example, the next two pending reason expressions are equivalent when used to match the pending reason text \texttt{Not enough job sslot(s)}:

**Example**

- Not enough
- Not enough \([a-zA-Z]\)* sslot(s)

The workload manager statistics utility can be used to find out what custom pending reason texts there are, and to add them to the pending reasons file. To do this, some test job can be forced to have such a pending reason, and the output of the job statistics utility can then be copy-pasted. For example, LSF shows custom pending reasons that look like this:

**Example**

Customized pending reason number <\texttt{integer}>

Here, \texttt{<integer>} is an identifier (an unsigned integer) for the pending reason, as defined by the administrator.

### 8.4.3 Locations

Sometimes it makes sense to restrict the workload manager to run jobs only on a defined subset of nodes. For example, if a user submits a multi-node job, then it is typically better to run all the job processes either on the on-premises nodes, or on the cloud nodes. That is, without mixing the node types used for the job. The \texttt{locations} feature of \texttt{cm-scale} allows this kind of restriction for HPC workload managers.

The \texttt{cm-scale} configuration allows one of these two modes to be selected:

1. \textit{forced location}: when the workload is forced to use one of the locations chosen by \texttt{cm-scale},
2. \textit{unforced location}: when workloads are free to run on any of the compute nodes that are already managed (running, freed or started) by \texttt{cm-scale}. This is the default if Auto Scaler is set up.

In NVIDIA Bright Cluster Manager 9.2, for a forced location, \texttt{cm-scale} supports these two different locations:
1. local: on-premises nodes,
2. cloud: AWS instances (Chapter 3 of the Cloudbursting Manual) or Azure instances (Chapter 5 of the Cloudbursting Manual)

To restrict the WLM location—that is to choose a forced location—the mixlocations advanced setting in the scaleserver role for the node must be set to no.

Example

```
[bright92->device[bright92]->roles[scaleserver]->advancedsettings]% set mixlocations no
[bright92->device[bright92]->roles[scaleserver]->advancedsettings]% commit
```

The location is automatically configured by the cluster manager when the node is added to the workload manager. Details per workload manager are described next.

**Slurm**

Slurm does not allow the assignment of node properties—features, in Slurm terminology—to jobs if no node exists that is labeled by this property. Thus any property used must be added to some node. This can be the template node if a dynamic resource provider is used, or it can be an appropriate off-premises node if a static resource provider is used. If the slurmclient role is assigned to a node—for example, a template node—then the location value for this node is automatically configured by the cluster manager.

The current location value can be found using the scontrol command. For example, for node001:

Example

```
[root@bright92 ]# module load slurm
[root@bright92 ]# scontrol show node node001 | grep AvailableFeatures
```

**PBS**

A new generic resource, resources_available.location, lets the administrator decide the locations where cm-scale can run PBS jobs.

If the pbsproclient role is assigned to a node, then the location value for this node is automatically configured by the cluster manager.

The current location value for a node can be found using the qmgr command. For example, for node001:

Example

```
[root@bright92 ]# module load openpbs
[root@bright92 ]# qmgr -c "print node node001" | grep location
set node node001 resources_available.location = local
```

**UGE**

A new generic resource, location, lets the administrator use cm-scale to restrict UGE jobs. The resource is added automatically as a string type complex attribute. The exec host locations can be found with the qhost command.

If the ugeclient role is assigned to a node, then the location value for this node is automatically configured by the cluster manager.

The current location value for a node can be found using the qhost -F command. For example, for node001:

Example

```
[root@bright92 ]# module load uge
[root@bright92 ]# qhost -F location -h node001
```

```
HOSTNAME    ARCH    NCPU    NSOC    NTHR    NLOAD    MEMTOT   MEMUSE   SWAPTO   SWAPUS
---------------------------------------------------------
global          -      -       -       -       -       -        -        -        -
node001  lx-amd64    2       2       2       2.01     3.9G     543.2M   0.0       0.0
hf:location=local
```
LSF

In order to allow cm-scale to restrict LSF jobs, the cluster manager configures a generic resource called `location` per node. The resource is added as a string resource in lsf.cluster.<CLUSTER_NAME: > configuration file.

The location value for this node is automatically configured by the cluster manager.

To verify that the resource is added, the `lshosts -s` command can be run:

Example

```
[root@bright92 ~]# lshosts -s location | head -1; lshosts -s location | grep node001
RESOURCE VALUE LOCATION
location local node001.cm.cluster
```

8.4.4 Azure Storage Accounts Assignment

If an Azure node is cloned manually from some node or node template, then the Azure node gets the same storage account as the node it has been cloned from. This may slow the nodes down if too many nodes use the same storage account. The cm-scale utility can therefore assign different storage accounts to nodes that are cloned like this.

The maximum number of nodes for such a storage account is defined by the `AZURE_DISK_ACCOUNT_NODES` parameter. This parameter has a value of 20 by default, and can be changed in the configuration file `/cm/local/apps/cm-scale/lib/python3.9/site-packages/cm-scale/config.py`. The cm-scale utility must be restarted after the change.

The newly-cloned-by-cm-scale Azure node gets a randomly-generated storage account name if other storage accounts already have enough nodes associated with them. That is, if other storage accounts have `AZURE_DISK_ACCOUNT_NODES` or more nodes.

The storage account name is assigned in the node cloud settings in storage submode. For example, in cmsh, the assigned storage accounts can be viewed as follows:

Example

```
[root@bright92 ~]# cmsh
[bright92]% device use cnode001
[bright92->device[cnode001]]% cloudsettings
[bright92->device[cnode001]->cloudsettings]% storage
[bright92->device[cnode001]->cloudsettings->storage]% get root-disk storageaccountname azurepaclogzjus1
[bright92->device[cnode001]->cloudsettings->storage]% get node-installer-disk storageaccountname azurepaclogzjus1
[bright92->device[cnode001]->cloudsettings->storage]% get bootdiagnosticsstorageaccountname azurepaclogzjus1
```

If a node is terminated and removed from the Bright configuration, then the storage account remains in Azure. It has to be explicitly manually removed by the administrator.

8.4.5 Mapping HPC Jobs To Particular Nodes

By default, cm-scale assumes that an HPC job submitted to a particular queue can take a node from outside the queue. This is because by assigning a category, or moving the node to a configuration overlay, the node will be moved to the appropriate queue eventually. From this point of view, the nodes form a single resource pool, and the nodes in the pool are re-purposed on demand.

In some scenarios there is a need for certain types of HPC jobs run only on particular types of nodes, without the nodes being re-purposed. A typical example: jobs with GPU code require cloud nodes that
have access to GPU accelerators, while jobs that do not have GPU code can use the less expensive non-GPU cloud nodes. For this case then, the GPU cloud node is started when the GPU job requires a node, and otherwise a non-GPU node is started.

Job segregation is achieved in cm-scale as follows:

1. The Pin Queues setting, which is an advanced setting in the scaleserver role for the node, is enabled:

   
   ```bash
   [bright92->device[bright92]->roles[scaleserver]->advancedsettings]% set pinqueues yes
   [bright92->device[bright92*]->roles[scaleserver*]->advancedsettings*]% commit
   ```

2. A new queue is created, or an existing one is used. The queue is used for the jobs that require a particular node type.

3. The particular node type is added to this queue. If the node is already defined in the cluster manager, then the administrator can assign the queue to the node in the workload manager client role. For example, if the workload manager is Slurm, then the queue is assigned to the nodes in the slurmclient role. If the node has not been defined yet and will be cloned on demand (according to the dynamic resource provider settings, page 383), then its template node is assigned to the queue. When a new node is cloned from the template, the queue is then inherited from the template node.

4. The previous two steps are repeated for each job type.

After that, if a user submits a job to one of the queues, then cm-scale starts or clones a node that is linked with the job queue.

The following cmsh session snippet shows a configuration example:

**Example**

```
[root@bright92 ~]# cmsh
[bright92]# device roles master
[bright92->device[bright92]->roles]% use scaleserver
[bright92->...roles[scaleserver]]% resourceproviders
[bright92->...roles[scaleserver]->resourceproviders]% add dynamic rp1
[bright92->...roles[scaleserver]->resourceproviders*] % set templatenode tnode1
[bright92->...roles[scaleserver]->resourceproviders*[rp1*]]% set noderange cnode001..cnode100
[bright92->...roles[scaleserver]->resourceproviders*[rp1*]]% commit
[bright92->...roles[scaleserver]->resourceproviders[rp1]]% clone rp2
[bright92->...roles[scaleserver]->resourceproviders*[rp2*]]% set templatenode tnode2
[bright92->...roles[scaleserver]->resourceproviders*[rp2*]]% set noderange cnode101..cnode200
[bright92->...roles[scaleserver]->resourceproviders*[rp2*]]% commit
[bright92->...roles[scaleserver]->resourceproviders[rp2]]% ...;
[bright92->...roles[scaleserver]]% engines
[bright92->...roles[scaleserver]->engines]% add hpc s1
[bright92->...roles[scaleserver]->engines*[e1*]]% set workloadmanager slurm
[bright92->...roles[scaleserver]->engines*[e1*]->trackers]% add queue tr1
[bright92->...roles[scaleserver]->engines*[e1*]->trackers*[tr1*]]% set queue q1
[bright92->...roles[scaleserver]->engines*[e1*]->trackers*[tr1*]]% commit
[bright92->...roles[scaleserver]->engines*[e1*]->trackers*[tr1*]]% clone tr2
[bright92->...roles[scaleserver]->engines*[e1*]->trackers*[tr2*]]% set queue q2
[bright92->...roles[scaleserver]->engines*[e1*]->trackers*[tr2*]]% commit
[bright92->...roles[scaleserver]->engines*[e1*]->trackers*[tr2*]]% category
[bright92->category]% clone default cat1
[bright92->category*[cat1*]]% roles
[bright92->category*[cat1*]->roles]% assign slurmclient
```
Further cm-scale Configuration And Examples

Using the preceding configuration, the user may submit a job with a regular workload manager submission utility specifying the queue q1 or q2, depending on whether the job requires nodes that should be cloned from tnode1 or from tnode2.

8.4.6 How To Exclude Unused Nodes From Being Stopped

If a node is idle, then by default cm-scale automatically stops or terminates the node. However, in some cases there may be a need to start a node on demand, and when it becomes idle, there may be a need to keep the node running. This can be useful if the administrator would like to investigate the performance of an application, or to debug some issues. After completing the investigation or debug session, the administrator can stop the node manually.

The parameter KEEP_RUNNING_RANGES keeps such nodes from being stopped or terminated. The parameter should be added to the configuration file /cm/local/apps/cm-scale/lib/python3.9/site-packages/cmscale/config.py. To have the changed setting take effect, the cm-scale service must be restarted.

KEEP_RUNNING_RANGES defines a map of resource provider names to node name ranges. Extra nodes can be added to the range of the nodes. However, if the extra node must not be stopped or terminated by cm-scale, then for each resource provider that has such an extra node, the value of extranodestop must be set to yes.

In the following example, nodes cnode002, cnode003, cnode004, and cnode010, are associated with the azurenodes1 resource provider. They are therefore never stopped or terminated by cm-scale. They are only started on demand by cm-scale.

The nodes cnode012 and cnode014 are associated with the azurenodes2 resource provider. They are therefore also not stopped or terminated by cm-scale.

Example

```python
opts = {
    "KEEP_RUNNING_RANGES": {
        "azurenodes1": "cnode002..cnode004,cnode010",
        "azurenodes2": "cnode012,cnode014"
    }
}
```

8.4.7 Prolog And Epilog Scripts With Auto Scaler

Sometimes the administrator would like some actions to be performed for a workload when the Auto Scaler allocates and starts using a node, or when the Auto Scaler deallocates and stops using a node. The administrator can arrange such actions by configuring prolog and epilog scripts (section 7.3.4) in the resource provider. The scripts are then executed on the nodes running the Auto Scaler service, i.e. with the ScaleServer role.

Both the dynamic and the static resource providers (section 8.2.2) support the following options:
1. allocationProlog: path to a shell script that is executed just before a node is started up by Auto Scaler

2. allocationEpilog: path to a shell script that is executed just before a node is powered off by Auto Scaler

3. allocationScriptsTimeout: the prolog and epilog scripts timeout (the script that is running is killed if the timeout is exceeded).

The prolog script runs when an existing node is about to start, and also runs when a node has just been cloned and is also about start.

The epilog script runs when a node is stopped or when a cloud node is terminated.

The prolog and epilog scripts are run per node, and can run in parallel. Thus if synchronization between them is needed, then it should be implemented by the scripts themselves.

The standard output and error messages of the executed scripts are mixed and added to the Auto Scaler as `debug2` log messages (the `debug2` logs can be enabled in the AdvancedSettings submode of the ScaleServer role). It therefore makes sense to keep the output reasonably small, informative, and human readable.

When the scripts are run, Auto Scaler passes environment variables that can be used inside the scripts in order to decide what to do. These environment variables are:

1. AS_NODE: node short hostname which the script started for;

2. AS_SCRIPT_TYPE: either "epilog" or "prolog";

3. AS_RESOURCE_PROVIDER: name of the resource provider where this script is configured;

4. AS_ENGINE: workload engine name, which workload requires the node ("unknown" if no workload requires the node).

By default the scripts are not defined, and therefore nothing is executed by default when nodes are stopped, terminated or started.

8.4.8 Queue Node Placeholders

A queue node placeholder is a node that does not yet exist, but has a corresponding object that exists, and the object has queues defined, amongst other properties. It can be used to plan resource use.

Job Rejection For Exceeding Total Cluster Resources

At the time of job submission, the workload manager checks the total available number of slots (used and unused) in a queue. This is the sum of the available slots (used and unused) provided by each node in that queue.

- Jobs that require less than the total number of slots are normally made to wait until more slots become available.

- Jobs that require more than this total number of slots are normally rejected outright by the workload manager, without being put into a wait state. This is because workload managers normally follow a logic that relies on the assumption that if the job demands more slots than can exist on the cluster as it is configured at present, then the cluster will never have enough slots to allow a job to run.

Assuming The Resources Can Never Be Provided

The latter assumption, that a cluster will never have enough slots to allow a job to run, is not true when the number of slots is dynamic, as is the case when cm-scale is used. When cm-scale starts up nodes, it adds them to a job queue, and the workload manager is automatically configured to allow users to submit jobs to the enlarged queue. That is, the newly available slots are configured as soon as possible.
so that waiting jobs are dealt with as soon as possible. For jobs that have already been rejected, and are not waiting, this is irrelevant, and users would have to submit the jobs once again.

Ideally, in this case, the workload manager should be configured to know about the number of nodes and slots that can be started up in the future, even if they do not exist yet. Based on that, jobs that would normally be rejected, could then also get told to wait until the resources are available, if it turns out that configured future resources will be enough to run the job.

**Slurm Resources Planning With Placeholders:**
Slurm allows nodes that do not exist yet to be defined. These are nodes with hostnames that do not resolve, and have the Slurm setting of `state=CLOUD` for cloud nodes, and `state=FUTURE` for other nodes. The cluster manager allows Slurm to add such “fake” nodes to Slurm queues dynamically, when not enough real nodes have yet been added. the cluster manager supports this feature only for Slurm at present.

This feature is not yet implemented for the other workload managers because they require the hostname of nodes that have been added to the workload manager configuration to be resolved.

Within the Slurm WLM instance it is possible to set a list of placeholder objects. In cmsh this can be done within the main `wlm` mode, selecting the Slurm instance, and then going into the `placeholders` submode. Each placeholder allows the following values to be set:

- **queue**: the queue name, used as key
- **maxnodes**: the maximum number of nodes that this queue allows
- **basenodename**: the base node name that is used when a new node name is generated
- **templatenode**: a template node that is used to provide user properties taken from its `slurmclient` role when new fake nodes are added.

For example, the following cmsh session uses the head node with an existing slurm instance to illustrate how the Slurm queue `defq` could be configured so that it always has a maximum of 32 nodes, with the nodes being like `node001`:

**Example**

```
[root@bright92 ~]# scontrol show part defq | grep " Nodes="
Nodes=node001
[root@bright92 ~]# cmsh
[bright92]# wlm use slurm
[bright92-vlmslurm]# placeholders
[bright92-vlmslurm]# add defq
[bright92-vlmslurm]# placeholders[defq]# set maxnodes 32
[bright92-vlmslurm]# placeholders[defq]# set basenodename placeholder
[bright92-vlmslurm]# placeholders[defq]# set templatenode node001
[bright92-vlmslurm]# placeholders[defq]# commit
[bright92-vlmslurm]# placeholders[defq]#
[root@bright92 ~]# scontrol show part defq | grep " Nodes="
Nodes=node001,placeholder[01-31]
```

If a new real node is added to the queue, then the number of placeholder nodes is decreased by one. The placeholders can also be configured in Bright View via the HPC resource, using the clickpath:

HPC→Workload Management Clusters→<Slurm instance>→JUMP TO Placeholders

**Preventing slurmctld From Restarting:**
If the number of nodes is changed, or if their names are changed in slurm.conf, then CMDaemon restarts the Slurm server daemon, `slurmctld`, to apply the changes. If the administrator needs to prevent `slurmctld` from restarting each time that a new node is added to Slurm, then the nodes can be added, or
cloned, to the cluster manager configuration manually, even if they do not have any IP address assigned yet. This is assuming that they get their IP addresses assigned over DHCP later on.

If the nodes are added to the configuration manually, then CMDaemon restarts `slurmctld` only once. This means that, when `cm-scale` starts the nodes, CMDaemon does not restart `slurmctld`. 
Containerization is a technology that allows processes to be isolated by combining cgroups, Linux namespaces, and (container) images.

- Cgroups are described in section 7.10.
- Linux namespaces represent independent spaces for different operating system facilities: process IDs, network interfaces, mount points, inter-process communication resources and others. Such cgroups and namespaces allow processes to be isolated from each other by separating the available resources as much as possible.
- A container image is a component of a container, and is a file that contains one or several layers. The layers cannot be altered as far the container is concerned, and a snapshot of the image can be used for other containers. A union file system is used to combine these layers into a single image. Union file systems allow files and directories of separate file systems to be transparently overlaid, forming a single coherent file system.

Cgroups, namespaces and image are the basis of a container. When the container is created, then a new process can be started within the container. Containerized processes running on a single machine all share the same operating system kernel, so they start instantly. No process is allowed to change the layers of the image. All changes are applied on a temporary layer created on top of the image, and these changes destroyed when the container is removed.

There are several ways to manage the containers, but the most powerful approaches use Docker, also known as Docker Engine (section 9.1), and Kubernetes (section 9.3). Docker manages containers on individual hosts, while Kubernetes manages containers across a cluster. The cluster manager integrates both of these solutions, so that setup, configuration and monitoring of containers becomes an easily-managed part of the cluster manager.

Singularity (section 9.7) is an application containerization tool, designed to execute containers as if they are just native applications on a host computer, and to work with HPC.

OpenShift (section 9.8) is Red Hat’s container manager.

### 9.1 Docker Engine

Docker integration with NVIDIA Bright Cluster Manager 9.2 for Docker version 20.10.10 is available at the time of writing of this paragraph (March 2022) on the x86_64 architecture for all the Bright-supported Linux distributions. For a more up-to-date status, the features matrix at [https://support.brightcomputing.com/feature-matrix/](https://support.brightcomputing.com/feature-matrix/) can be checked.

Docker Engine (or just Docker) is a tool for container management. Docker allows containers and their images to be created, controlled, and monitored on a host using Docker command line tools or the Docker API.
Swarm mode, which allows containers to spawn on several hosts, is not formally supported by NVIDIA Bright Cluster Manager 9.2. This is because NVIDIA Bright Cluster Manager 9.2 provides Kubernetes for this purpose instead.

Docker provides a utility called `docker`, and two daemons called `containerd` (the default provided by the cluster manager), and `dockerd`. Additional functionality includes pulling the container image from a specific image registry (section 9.2), configuring the container network, setting `systemd` limits, and attaching volumes.

### 9.1.1 Docker Setup

The cluster manager provides the `cm-docker` package. The package includes the following components:

- Docker itself, that provides an API and delegates the container management to Containerd;
- Containerd runtime, that manages OCI images and OCI containers (via runC);
- runC, a CLI tool for spawning and running containers according to the OCI specification runtime;
- docker-py, a Python library for the Docker API.

Typically, however, the administrator is expected to simply run the `cm-docker-setup` utility, which is provided by the cluster manager’s `cm-setup` package. Running `cm-docker-setup` takes care of the installation of the `cm-docker` package and also takes care of Docker setup. If run without options the utility starts up an terminal GUI dialog (figure 9.1).

![Figure 9.1: cm-docker-setup Ncurses startup](image)

The `cm-docker-setup` utility asks several questions, such as which Docker registries are to be used, what nodes Docker is to be installed on, whether the Nvidia container runtime should be installed, and so on. If `cm-docker-setup` is used with the `-c` option, and given a YAML configuration file `<YAMLfile>`, then a runtime configuration is loaded from that file. The YAML file is typically generated and saved from an earlier run.

When the questions in the terminal GUI dialog have been answered and the deployment is carried out, the utility:

- installs the `cm-docker` package, if it has not been installed yet
- then assigns the DockerHost role to the node categories or head nodes that were specified
- adds health checks to the cluster manager monitoring configuration
- performs the initial configuration of Docker.

The regular nodes on which Docker is to run, are restarted by the utility, if needed. The restart operation provisions the updated images from the image directory onto the nodes.

The `cm-docker` package also includes a module (section 2.2) file, `docker`, which must be loaded in order to use the `docker` command. By default only the administrator can run the `docker` commands after setup (some output ellipsized):
Example

[root@bright92 ~]# ssh node001
[root@node001 ~]# module load docker
[root@node001 ~]# docker info
Containers: 0
Images: 0
...  
Docker Root Dir: /var/lib/docker
Debug Mode: false
Registry: https://index.docker.io/v1/
Labels:
Experimental: false
Insecure Registries:  
127.0.0.0/8
Registry Mirrors:  
https://harbor-proxy.brightcomputing.com/
Live Restore Enabled: false
[root@node001 ~]#  

and the hello-world image can be run as usual with:

Example

[root@node001 ~]# docker run hello-world
Unable to find image 'hello-world:latest' locally
latest: Pulling from library/hello-world
2db29710123e: Pull complete
Digest: sha256:cc15c5b292d8525effc0f89cb299f1804f3a725c8d05e158653a563f15e4f685
Status: Downloaded newer image for hello-world:latest
Hello from Docker!
This message shows that your installation appears to be working correctly.
...

Or, for example, importing and running Apache containers with Docker may result in the following output:

Example

[root@node001 ~]# module load docker
[root@node001 ~]# docker run httpd & docker run httpd &
... runs a couple of Apache containers...
[root@node001 ~]# docker container ls
CONTAINER ID IMAGE COMMAND CREATED STATUS PORTS NAMES
acdbe2f3667b httpd "httpd-foreground" 13 seconds ago Up 11 seconds 80/tcp quizzical_bhabha
64787a8524dd httpd "httpd-foreground" 13 seconds ago Up 11 seconds 80/tcp funny_hypatia
...
[root@node001 ~]#  

Using Docker directly means being root on the host. It is rarely sensible to carry out regular user actions as the root user at all times.

So, to make Docker available to regular users, Kubernetes provides a user management layer and restrictions.

After Docker has been installed, Kubernetes can be set up to allow regular user access to the Docker containers as covered in section 9.3. It is a best practice for regular users to use Kubernetes instead of Docker commands directly.
9.1.2 Integration With Workload Managers
The cluster manager does not provide integration of Docker with workload managers. The administrator can however tune the workload managers in some cases to enable Docker support.

- LSF – An open beta version of LSF with Docker support is available from the IBM web site. This LSF version allows jobs to run in Docker containers, and monitors the container resources per job.
- PBS Pro – Altair provides a hook script that allows jobs to start in Docker containers. Altair should be contacted to obtain the script and instructions.

9.1.3 DockerHost Role
When cm-docker-setup is executed, the DockerHost role is assigned to nodes or categories. The DockerHost role is responsible for Docker service management and configuration.

From cmsh, the configuration parameters can be managed from the Docker::Host role:

Example

```
[root@bright92 ~]# cmsh
[bright92]% category use default
[bright92->category[default]]% roles
[bright92->category[default]->roles] assign docker::host
[bright92->category[default]->roles[Docker::Host]]% show
```

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>Name</td>
<td>Docker::Host</td>
</tr>
<tr>
<td>Revision</td>
<td>DockerHostRole</td>
</tr>
<tr>
<td>Add services</td>
<td>yes</td>
</tr>
<tr>
<td>Provisioning associations</td>
<td>&lt;0 internally used&gt;</td>
</tr>
<tr>
<td>Spool</td>
<td>/var/lib/docker</td>
</tr>
<tr>
<td>Temp dir</td>
<td>$spool/tmp</td>
</tr>
<tr>
<td>Enable SELinux</td>
<td>yes</td>
</tr>
<tr>
<td>Default Ulimits</td>
<td></td>
</tr>
<tr>
<td>Debug</td>
<td>no</td>
</tr>
<tr>
<td>Log Level</td>
<td>info</td>
</tr>
<tr>
<td>Bridge IP</td>
<td></td>
</tr>
<tr>
<td>Bridge MTU</td>
<td>0</td>
</tr>
<tr>
<td>API Sockets</td>
<td>unix:///var/run/docker.sock</td>
</tr>
<tr>
<td>Iptables</td>
<td>yes</td>
</tr>
<tr>
<td>User Namespace Remap</td>
<td></td>
</tr>
<tr>
<td>Insecure Registries</td>
<td></td>
</tr>
<tr>
<td>Enable TLS</td>
<td>no</td>
</tr>
<tr>
<td>Verify TLS</td>
<td>no</td>
</tr>
<tr>
<td>TLS CA</td>
<td></td>
</tr>
<tr>
<td>TLS Certificate</td>
<td></td>
</tr>
<tr>
<td>TLS Key</td>
<td></td>
</tr>
<tr>
<td>Certificates Path</td>
<td>/etc/docker</td>
</tr>
<tr>
<td>Storage Backends</td>
<td>&lt;0 in submode&gt;</td>
</tr>
<tr>
<td>Containerd Socket</td>
<td></td>
</tr>
<tr>
<td>Runtime</td>
<td>runc</td>
</tr>
<tr>
<td>Options</td>
<td></td>
</tr>
</tbody>
</table>

The Docker host parameters that CMDaemon can configure in the DockerHost role, along with a description, are shown in table 9.1:
<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Add services*</td>
<td>Add services to nodes belonging to this node. Care must be taken if setting this to no. (default: yes)</td>
</tr>
<tr>
<td>Spool</td>
<td>Root of the Docker runtime (default: /var/lib/docker)</td>
</tr>
<tr>
<td>Tmp dir</td>
<td>Location for temporary files. Default: $&lt;spool&gt;/tmp, where $&lt;spool&gt; is replaced by the path to the Docker runtime root directory</td>
</tr>
<tr>
<td>Enable SELinux*</td>
<td>Enable selinux support in Docker daemon (default: yes)</td>
</tr>
<tr>
<td>Default Ulimits</td>
<td>Set the default ulimit options for all containers</td>
</tr>
<tr>
<td>Debug*</td>
<td>Enable debug mode (default: no)</td>
</tr>
<tr>
<td>Log Level</td>
<td>Set the daemon logging level. In order of increasing verbosity: fatal, error, warn, info, debug. (default: info)</td>
</tr>
<tr>
<td>Bridge IP</td>
<td>Network bridge IP (not defined by default)</td>
</tr>
<tr>
<td>Bridge</td>
<td>Attach containers to a network bridge (not defined by default)</td>
</tr>
<tr>
<td>MTU</td>
<td>Set the containers network MTU, in bytes (default: 0, which does not set the MTU at all)</td>
</tr>
<tr>
<td>API Sockets</td>
<td>Daemon socket(s) to connect to (default: unix://var/run/docker.sock)</td>
</tr>
<tr>
<td>Iptables*</td>
<td>Enable iptables rules (default: yes)</td>
</tr>
<tr>
<td>User Namespace Remap</td>
<td>User/Group setting for user namespaces (not defined by default). It can be set to any of &lt;UID&gt;, <a href="">UID:GID</a>, &lt;username&gt;, <a href="">username:groupname</a>. If it is used, then user_namespace.enable=1 must be set in the kernel options for the relevant nodes, and those nodes must be rebooted to pick up the new option.</td>
</tr>
</tbody>
</table>

...continues
...continued

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Insecure Registries</td>
<td>If registry access uses HTTPS but does not have proper certificates distributed, then the administrator can make Docker accept this situation by adding the registry to this list (empty by default)</td>
</tr>
<tr>
<td>Enable TLS*</td>
<td>Use TLS (default: no)</td>
</tr>
<tr>
<td>Verify TLS*</td>
<td>Use TLS and verify the remote (default: no)</td>
</tr>
<tr>
<td>TLS CA</td>
<td>Trust only certificates that are signed by this CA (not defined by default)</td>
</tr>
<tr>
<td>TLS Certificate</td>
<td>Path to TLS certificate file (not defined by default)</td>
</tr>
<tr>
<td>TLS Key</td>
<td>Path to TLS key file (not defined by default)</td>
</tr>
<tr>
<td>Certificates Path</td>
<td>Path to Docker certificates (default: /etc/docker)</td>
</tr>
<tr>
<td>Storage Backends</td>
<td>Docker storage back ends. Storage types can be created and managed, in a submode under this mode. The available types are described in table 9.2. Each of these storage types has options that can be set from within the submode.</td>
</tr>
<tr>
<td>Containerd Socket</td>
<td>Path to the containerd socket (default: not used)</td>
</tr>
<tr>
<td>Runtime</td>
<td>Docker runtime</td>
</tr>
<tr>
<td>Options</td>
<td>Additional parameters for docker daemon</td>
</tr>
</tbody>
</table>

* Boolean (takes yes or no as a value)

Table 9.1: Docker::Host role options

9.1.4 Iptables

By default iptables rules have been added to nodes that function as a Docker host, to let network traffic go from the containers to outside the pods network. If this conflicts with other software that uses iptables, then this option can be disabled. For example, if the docker::host role has already been assigned to the nodes via the default category, then the iptables rules that are set can be disabled by setting the iptables parameter in the Docker::Host role to no:

Example

```
[root@bright92 ~]# cmsh
[bright92]# category use default
[bright92->category[default]]% roles
[bright92->category[default]->roles]% use docker::host
[bright92->category[default]->roles[Docker::Host]]% set iptables no
[bright92->category[default*]->roles*[Docker::Host*]]% commit
```
9.1.5 Storage Backends

A core part of the Docker model is the efficient use of containers based on layered images. To implement this, Docker provides different storage back ends, also called storage drivers. These storage back ends rely heavily on various filesystem features in the kernel or volume manager. Some storage back ends perform better than others, depending on the circumstances.

The default storage back end configured by cm-docker-setup is overlay2. Storage back ends supported by Docker are listed in table 9.2:

<table>
<thead>
<tr>
<th>Technology</th>
<th>Description</th>
<th>Backend Name</th>
</tr>
</thead>
<tbody>
<tr>
<td>OverlayFS</td>
<td>This is a modern union filesystem. It is the preferred storage driver for recent Docker versions. It has been in the mainline Linux kernel since version 3.18, with additional improvements for Docker in version 4.0. All of the distributions that the cluster manager supports have backported the kernel changes needed for this to work.</td>
<td>overlay2</td>
</tr>
<tr>
<td>Device Mapper</td>
<td>Deprecated since Docker Engine 18.09. It is a kernel-based framework that has been included in the mainline Linux kernel since version 2.6.9. It underpins many advanced volume management technologies on Linux. The driver stores every image and snapshot on its own virtual device, and works at the block level rather than the file level. • A loopback mechanism can be implemented using loop-lvm mode. This allows files on a local disk to be managed as if they are on a physical disk or block device. This is simpler than the thin pool mode, but is strongly discouraged for production use. In the cluster manager this mode is implemented by selecting the option loopback (testing only). This is selected in the storage back end selection screen of the cm-docker-setup installation. • A thin pool mode can be implemented using direct-lvm mode. This uses a logical volume as a thin pool to use as backing for the storage pool, and uses a spare block device. Configuring this is normally more involved. In the cluster manager this mode is implemented by selecting the option block (production ready). This is selected in the storage back end selection screen of the cm-docker-setup installation session.</td>
<td>devicemapper</td>
</tr>
</tbody>
</table>

Device mapper options for these modes are described in Table 9.3...continues
Continued

<table>
<thead>
<tr>
<th>Technology</th>
<th>Description</th>
<th>Backend Name</th>
</tr>
</thead>
<tbody>
<tr>
<td>AUFS</td>
<td>This was the first storage back end that Docker used. AUFS is not included in the mainline Linux kernel. Out of the distributions that NVIDIA Bright Cluster Manager 8.2 supports, it is only Ubuntu that supports it.</td>
<td>aufs</td>
</tr>
</tbody>
</table>

Table 9.2: Docker storage back ends

The `docker info` command, amongst many other items, shows the storage driver and related settings that are being used in Docker:

**Example**

```
[root@bright92 ~]# module load docker
[root@bright92 ~]# docker info

Client:
  Context:  default
  Debug Mode: false

Server:
  Containers: 18
  Running: 8
  Paused: 6
  Stopped: 4
  Images: 1
  Server Version: 20.10.9
  Storage Driver: overlay2
  Backing Filesystem: xfs
  Supports d_type: true
  Native Overlay Diff: true
  userxattr: false
  Logging Driver: json-file
  Cgroup Driver: cgroupfs
  Cgroup Version: 1
  Plugins:
    Volume: local
    Network: bridge host ipvlan macvlan null overlay
    Log: awslogs fluentd gcplogs gelf journald json-file local logentries splunk syslog ...
```

Docker data volumes are not controlled by the storage driver. Reads and writes to data volumes bypass the storage driver. It is possible to mount any number of data volumes into a container. Multiple containers can also share one or more data volumes.

More information about Docker storage back ends is available at [https://docs.docker.com/engine/userguide/storagedriver](https://docs.docker.com/engine/userguide/storagedriver).

**Device Mapper Driver Settings Support**

The cluster manager supports device mapper driver settings more explicitly than the other driver back end settings.

By default the device mapper storage back end is added automatically, and can be configured in the `storagebackends` submode of the DockerHost role:

**Example**
9.1 Docker Engine

```
[d92->device[d92->roles[d92::host]]% storagebackends
d92->device[d92->roles[d92::host]]% use devicemapper
d92->device[d92->roles[d92::host]]% show
```

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>Name</td>
<td>devicemapper</td>
</tr>
<tr>
<td>Revision</td>
<td>devicemapper</td>
</tr>
<tr>
<td>Type</td>
<td>devicemapper</td>
</tr>
<tr>
<td>Loop Data Size</td>
<td>xfs</td>
</tr>
<tr>
<td>Loop Metadata Size</td>
<td>64K</td>
</tr>
<tr>
<td>Loop Device Size</td>
<td>yes</td>
</tr>
<tr>
<td>Pool Device</td>
<td></td>
</tr>
<tr>
<td>Filesystem</td>
<td>xfs</td>
</tr>
<tr>
<td>Block Size</td>
<td>64K</td>
</tr>
<tr>
<td>Blk Discard</td>
<td>yes</td>
</tr>
<tr>
<td>Mkfs Arguments</td>
<td></td>
</tr>
<tr>
<td>Mount Options</td>
<td></td>
</tr>
</tbody>
</table>

The parameters that are used in the Docker device mapper back end are described in table 9.3:

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
<th>Option to docker</th>
</tr>
</thead>
<tbody>
<tr>
<td>Blk Discard^</td>
<td>Enables or disables the use of blkdiscard when removing device mapper devices (default: yes)</td>
<td>dm.blkdiscard</td>
</tr>
<tr>
<td>Block Size</td>
<td>Custom blocksize to use for the thin pool (default: 64kB)</td>
<td>dm.blocksize</td>
</tr>
<tr>
<td>Filesystem</td>
<td>Filesystem type to use for the base device (default: xfs)</td>
<td>dm.fs</td>
</tr>
<tr>
<td>Loop Data Size</td>
<td>Size to use when creating the loopback file for the data virtual device which is used for the thin pool (default: 100GB)</td>
<td>dm.loopdatasize</td>
</tr>
<tr>
<td>Loop Device Size</td>
<td>Size to use when creating the base device, which limits the size of images and container (not set by default)</td>
<td>dm.basesize</td>
</tr>
<tr>
<td>Loop Metadata Size</td>
<td>Size to use when creating the loopback file for the metadata device which is used for the thin pool (default: 2GB)</td>
<td>dm.loopmetadatasize</td>
</tr>
<tr>
<td>Mkfs Arguments</td>
<td>Extra mkfs arguments to be used when creating the base device</td>
<td>dm.mkfsarg</td>
</tr>
</tbody>
</table>

...continues
...continued

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
<th>Option to docker</th>
</tr>
</thead>
<tbody>
<tr>
<td>Mount Options</td>
<td>Extra mount options used when mounting the thin devices</td>
<td>dm.mountopt</td>
</tr>
<tr>
<td>Pool Device</td>
<td>Custom block storage device to use for the thin pool (not set by default)</td>
<td>dm.thinpooldev</td>
</tr>
</tbody>
</table>

* Boolean (takes yes or no as a value)

Table 9.3: Device mapper back end Docker options

For back end driver storage settings other than device mapper, such as AUFS or OverlayFS, settings can be added as options if needed. In cmsh this can be done by setting the options parameter in the storagebackend submodule under the docker::host role.

9.1.6 Docker Monitoring

When cm-docker-setup runs, it configures and runs the following Docker health checks:

1. makes a test API call to the endpoint of the Docker daemon
2. checks containers to see that none is in a dead state

The Docker daemon can be monitored outside of the cluster manager with the usual commands directly.

The cluster manager ways to manage or check on Docker include the following:

In CMDaemon, the docker service can be checked:

Example

```
[bright92->device[node001]->services]% list
Service (key) Monitored Autostart
------------------------ ---------- ----------
docker yes yes
ns1od yes yes

[bright92->device[node001]->services]% show docker
Parameter Value
--------------------------------------------------------------
Revision Service docker
Run if ALWAYS
Monitored yes
Autostart yes
Timeout -1
Belongs to role yes
Sickness check script
Sickness check script timeout 10
Sickness check interval 60
```

The docker0 interface statistics can be checked within the nodeoverview output:

Example

```
[bright92->device[node001]]% nodeoverview ...
Interface Received Transmitted
--------------------------- ---------------------------
```
9.1 Docker Engine

The measurable Docker checks if the docker service is healthy.

Example

```
[root@bright92 ~]# dumpmonitoringdata -1h now Docker
```

<table>
<thead>
<tr>
<th>Timestamp</th>
<th>Value</th>
<th>Info</th>
</tr>
</thead>
<tbody>
<tr>
<td>2021/11/29 11:52:44.146</td>
<td>PASS</td>
<td></td>
</tr>
<tr>
<td>2021/11/30 18:28:44.146</td>
<td>PASS</td>
<td></td>
</tr>
</tbody>
</table>

9.1.7 Docker Setup For NVIDIA

NVIDIA GPU Cloud (NGC) is a cloud platform that runs on NVIDIA GPUs. NGC containers are lightweight containers that run applications on NVIDIA GPUs. The applications are typically HPC, machine learning, or deep learning applications.

An NGC can be set up to run NGC containers from the registry http://ngc.nvidia.com.

Docker can be configured as an NGC running NGC containers by using the NVIDIA Container Toolkit.

The cluster manager package provided for this is: cm-nvidia-container-toolkit.

One way to install and deploy this package is as part of the Docker installation, when running cm-docker-setup (section 9.1.1), where the cluster administrator selects yes as the answer to the request: "Do you want to install the Nvidia Runtime for Docker?".

Alternatively, if Docker has already been installed via cm-docker-setup, and if the package has not been installed, then it can be installed via the package manager, yum or apt. The toolkit has to be running on the compute nodes that have GPUs, which means that the installation must go to the appropriate node image (section 12.4). For example, if the appropriate image is gpu-image:

Example

```
# yum install --installroot=/cm/images/gpu-image cm-nvidia-container-toolkit
```

The nodes that use that GPU image can then be rebooted to pick up the new package.

The GPU status can then be printed with the NVIDIA system management interface command. For example, if the image has been picked up by node001:

Example

```
[root@bright92 ~]# ssh node001
Last login: Thu Dec 2 09:24:03 2021 from bright92.cm.cluster
[root@node001 ~]# module load docker
[root@node001 ~]# docker run --runtime=nvidia --rm nvidia/cuda:11.4.0-base nvidia-smi
Unable to find image 'nvidia/cuda:11.4.0-base' locally
11.4.0-base: Pulling from nvidia/cuda
```

Digest: sha256:f0a5937399da5e4efb37fd7b75beb8c48b84dc381243c481fc5f9fcad42b66
Status: Downloaded newer image for nvidia/cuda:11.4.0-base
Mon Mar 7 17:30:48 2022

```
          +-----------------------------------------------------------------------------+
          | NVIDIA-SMI 440.33.01 Driver Version: 440.33.01 CUDA Version: 11.4             |
          |                                                                 +-----------------+-----------------+-----------------+          |
          | | GPU Name | Persistence-M| Bus-Id | Disp.A | Volatile Uncorr. ECC | | |
          | Fan Temp | Perf | Pwr:Usage/Cap | Memory-Usage | GPU-Util | Compute M. | |
          |-----------------+-----------------+-----------------+-----------------+-----------------+-----------------+ |
```
The available Cuda Docker images can be found at https://hub.docker.com/r/nvidia/cuda. How to run applications in an NGC with the cluster manager is illustrated for PyTorch, TensorFlow, and MXNet, in the white paper at https://www.brightcomputing.com/hubfs/docs/Bright-NGC-WPP-v01.pdf. The sessions illustrated in that paper require the installation of Kubernetes (section 9.3), or Singularity (section 9.7).

9.2 Docker Registries

When a user creates a new container, an image specified by the user should be used. The images are kept either locally on a host, or in a registry. The image registry can be provided by a cloud provider or locally.

9.2.1 Docker And Harbor Registries: Introduction

Docker Hub, A Remote Registry

By default, Docker searches for images in Docker Hub, which is a cloud-hosted public and private image registry. Docker Hub serves a huge collection of existing images that users can make use of. Every user is allowed to create a new account, and to upload and share images with other users. Using the Docker client, a user can search for already-published images, and then pull them down to a host in order to build containers from them.

When an image is found in the registry, the Docker client verifies if the latest version of the image has already been downloaded. If not, then it downloads the images, and stores them locally. It also tries to synchronize them when a new container is created. When the latest image is downloaded, Docker creates a container from the image layers that are formatted to be used by a union file system. Docker can make use of several union file system variants, including AUFS, btrfs, vfs, and DeviceMapper.

Local Image Registry Options: Classic Docker Registry Vs Harbor

Besides using Docker Hub for the image registry, the administrator can also install a local image registry on the cluster. The cluster manager provides two ways to integrate such a local registry with the cluster, based on existing open source projects:

- The first one is the classic docker registry provided by Docker Inc, and can be useful if the registry is used by trusted users.

- The second registry, Harbor, developed by VMware and introduced in NVIDIA Bright Cluster Manager version 8.1-5, provides additional features such as security and identity management, and is aimed at the enterprise.

Both options can be installed with the cm-container-registry-setup utility, which comes with the cluster manager’s cm-setup package.
9.2 Docker Registries

9.2.2 Docker And Harbor Registries: Setup And Configuration

Docker Registry and Harbor can be installed via the `cm-container-registry-setup` command-line utility. They can also be installed via Bright View in NVIDIA Bright Cluster Manager for versions beyond 8.1-6 as follows:

- The Docker Registry Deployment Wizard is launched via the clickpath:
  Containers → Docker → Docker Registry Wizard

- Either Docker Registry, or Harbor, should be chosen as a registry.

- A single node is ticked for the deployment. The address, port, and the root directory for storing the container images are configured. If the head node is selected for Harbor, then the setup later on asks to open the related port on the head node. This is to make Harbor and the Harbor UI externally accessible.

- In the summary page, if the `Ready for deployment` box is ticked, then the administrator can go ahead with deploying the registry.

- When the deployment is complete, the Docker Registry becomes ready for use. Harbor can take a few additional minutes to be ready for use.

Similar to the case of etcd nodes (section 9.3.2), nodes that run Harbor or Docker Registry have the `datanode` option (section 5.4.4) when installed by the cluster manager utilities. The option helps prevent the registry being wiped out by accident, and is added when the `cm-container-registry-setup` utility is used to install Harbor or Docker Registry. This extra protection is put into place because if a registry is wiped out, then the state of images in the container becomes incoherent and cannot be restored.

**Harbor UI** If the head node is where Harbor is to be installed, and is to be made externally accessible, then the Harbor UI can be accessed at `https://<head node hostname>:9443`.

If a different node is used for Harbor to be installed, then the related port must be forwarded locally. Harbor can be logged into by default with the `admin` user and the default `Harbor12345` password. It is recommended to change that password after the first login.

**Clair** Harbor comes with Clair, a tool for vulnerability static analysis of container images.

More information on using Clair with Harbor can be found at `https://github.com/vmware/harbor/blob/master/docs/user_guide.md#vulnerability-scanning-via-clair`.

**Dealing with a pre-existing Kubernetes or Harbor installation:** Since Harbor uses Docker internally, and because Kubernetes customizes Docker networking, it means that nodes where Kubernetes is running cannot be reused for Harbor, and that nodes where Harbor is running cannot be reused for Kubernetes.

**Docker Registry Daemon Configuration Using The Docker Registry Role**

The Docker Registry role is used to configure and manage the `docker-registry` daemon, and its parameters are described in Table 9.4:

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Domain</td>
<td>Main domain name (default: hostname of the node)</td>
</tr>
</tbody>
</table>

...continues
Parameter Description

Alt Domains Alternative domain names (default: FQDN of the node)

Port Port (default: 5000)

Spool Dir Spool directory (default: /var/lib/docker-registry)

* Boolean (takes yes or no as a value)

Table 9.4: Docker Registry role parameters

The values stored in the Docker Registry role are not supposed to be changed, but they are useful for the uninstall procedure, and also as a record of the settings for the administrator.

Further details on the docker-registry daemon can be found at https://github.com/docker/distribution.

Harbor Daemon Configuration Using The Harbor Role

The Harbor role is used to configure and manage the harbor daemon. The parameters of the role are described in table 9.5:

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Domain</td>
<td>Main domain name (default: hostname of the node)</td>
</tr>
<tr>
<td>Alt Domains</td>
<td>Alternative domain names (default: FQDN of the node)</td>
</tr>
<tr>
<td>Port</td>
<td>Port (default: 9443)</td>
</tr>
<tr>
<td>Spool Dir</td>
<td>Spool directory (default: /var/lib/harbor)</td>
</tr>
<tr>
<td>Default Password</td>
<td>Default password of the Harbor admin user (default: Harbor12345)</td>
</tr>
</tbody>
</table>

...continues
9.3 Kubernetes

Kubernetes is an open-source platform for automating deployment, scaling, and operations of application containers across clusters of hosts. With Kubernetes, it is possible to:

- scale applications on the fly
- seamlessly update running services
- optimize hardware usage by using only the resources that are needed

The cluster manager provides the administrator with the required packages, allows Kubernetes to be set up on cluster, and manages and monitors Kubernetes. More information about the design of Kubernetes, its command line interfaces, and other Kubernetes-specific details, can be found at the official online documentation at https://kubernetes.io/docs/.

Docker integration with NVIDIA Bright Cluster Manager 9.2 for Docker version 20.10.10 is available at the time of writing of this paragraph (March 2022) on the x86_64 architecture for all the Bright-supported Linux distributions. For a more up-to-date status, the features matrix at https://support.brightcomputing.com/feature-matrix/ can be checked.

### 9.3.1 Reference Architecture

A reference architecture for Kubernetes in the cluster manager comprises:

- **etcd nodes**: An etcd cluster—the Kubernetes distributed key-value storage—runs on regular nodes only, and uses an odd number (1, 3, 5 ...) of nodes.

---

### Table 9.5: Harbor role parameters

The values stored in the Harbor role are not supposed to be changed, but they are useful for the uninstall procedure, and also as reminder of the settings for the administrator.

```
[bright92->device[bright92]->roles[generic::harbor]]% environments
[bright92->device[bright92]->roles[generic::harbor]->environments]% list
```

<table>
<thead>
<tr>
<th>Name (key)</th>
<th>Value</th>
<th>Node Environment</th>
</tr>
</thead>
<tbody>
<tr>
<td>alt_domains</td>
<td>harbor_node01.cm.cluster</td>
<td>no</td>
</tr>
<tr>
<td>db_password</td>
<td>&lt;generated password&gt;</td>
<td>no</td>
</tr>
<tr>
<td>domain</td>
<td>node001</td>
<td>no</td>
</tr>
<tr>
<td>external_network</td>
<td>True</td>
<td>no</td>
</tr>
<tr>
<td>port</td>
<td>9443</td>
<td>no</td>
</tr>
<tr>
<td>redirect_port</td>
<td>65535</td>
<td>no</td>
</tr>
<tr>
<td>spool_dir</td>
<td>/var/lib/harbor</td>
<td>no</td>
</tr>
</tbody>
</table>

Further details on Harbor can be found at https://vmware.github.io/harbor.
• **master nodes**: Kubernetes master units run on head or compute nodes. 2 (or 3) are recommended for High Availability (HA). In a Bright HA configuration, both or none of the head nodes should be selected. That is, it must not run on only one head node of an HA configuration.

• **worker nodes**: Kubernetes worker units run on regular nodes only.

Since NVIDIA Bright Cluster Manager version 8.2, multiple clusters of Kubernetes can be deployed. In such a configuration the same node cannot be shared across different Kubernetes clusters.

A Kubernetes API server proxy based on NGINX runs on every node, except for on nodes that run etcd. The proxy also runs on the head node(s).

Because of the server proxy, a port is reserved on the head node(s) for every Kubernetes cluster. This is required for Kubernetes HA (section 9.3.1), and it also allows kubectl and other tools such as Helm to be used from the head node, to access each Kubernetes cluster.

**Kubernetes HA**

For a Kubernetes HA setup the minimum node requirements are:

• at least 3 nodes for Etcd

• at least 2 nodes for Kubernetes Master

In an average cluster, Bright Computing recommends 3 nodes for the etcd cluster, and 3 nodes for the Kubernetes master.

Even without a Kubernetes master in an HA configuration, there is no downtime for existing pods running on the worker nodes. The worker nodes will still continue to work.

However, Kubernetes HA is needed to be able to schedule tasks, spawn new pods, and in general keep the cluster in the desired state.

**9.3.2 Kubernetes Setup**

The cluster manager provides the following Kubernetes-related packages:

• The `cm-kubernetes-master, cm-kubernetes-node, conntrack` and `nginx` packages: These are always installed on the head node(s) and on the master and worker node(s).

• The `cm-etcd` package is installed on the nodes selected for etcd. In a similar way to the case of Harbor or Docker Registry (section 9.2.2), the nodes that run etcd are protected by the cluster manager with the datanode option (section 5.4.4). For etcd nodes, the option is added during the `cm-kubernetes-setup` installation. As in the case for the registries, the datanode option is set in order to help prevent the administrator from wiping out the existing state of etcd nodes. Wiping out the state of etcd nodes means that the Kubernetes cluster becomes incoherent and that it cannot be restored to where it was just before the etcd nodes were wiped. The etcd version installed by the cluster manager package is 3.5.0.

The packages are installed automatically from the repository when the administrator runs `cm-kubernetes-setup` from the command line.

The log file produced by the setup can be found in `/var/log/cm-kubernetes-setup.log`.

**Kubernetes Core Add-ons**

During setup, some critical add-on components such as CoreDNS and Calico are automatically deployed in the `kube-system` namespace. In the cluster manager all add-ons are treated as Kubernetes applications (section 9.4), and belong to the default app group `system`.

A `cmsh` treeview illustrating the hierarchy to access these applications is:
CoreDNS: The DNS server add-on for internal service discovery. It reads the IP addresses of services and pods from Etcd, and resolves domain names for them. If a domain name is not found because the domain is external to the Kubernetes cluster, then CoreDNS forwards the request to the main DNS server. The cluster manager uses CoreDNS version 1.62.

Calico: A modern SDN (Software-Defined Networking) add-on to manage a cluster-wide network for pods. Calico uses an agent called Felix to run on each node as a pod. The cluster manager uses Calico version 3.10.

When the Kubernetes cluster is composed of more than 50 nodes, the component Typha is also deployed for better scalability. The number of Typha replicas is calculated by allocating one Typha replica per 150 nodes, with a minimum of 3 (above 50 nodes) and a maximum of 20.

Further details on Calico can be found at https://docs.projectcalico.org/.

Kubernetes Optional Add-ons
The following add-ons are installed by default unless otherwise noted. However, the user can choose to skip some or all of them during the setup.

Kubernetes Dashboard: The web user interface add-on for GUI cluster administration and metrics visualization. The cluster manager uses Kubernetes Dashboard version 2.0.

There are two ways to access the dashboard:


- Users on an external network can log in to kubectl or Kubernetes Dashboard by following the procedures described in section 9.3.16.

Kubernetes Metrics Server: This add-on is a replacement for Heapster. It aggregates metrics, and provides container monitoring and performance analysis. It exposes metrics via an API. The cluster manager runs Metrics server version 1.0.0.

Helm: An add-on that manages charts, which are packages of pre-configured Kubernetes resources. The Helm component is installed and properly configured with the cluster manager’s Kubernetes installation by default. It is initialized and ready for use for every Kubernetes user when the Kubernetes module is loaded. The cluster manager uses Helm version 3.

NGINX Ingress Controller: The official Kubernetes ingress controller add-on is built around the Kubernetes Ingress resource, using a ConfigMap to store the NGINX configuration. Ingress provides HTTP and HTTPS routes from outside a Kubernetes cluster to services within the cluster. Traffic routing is controlled by rules defined in the Ingress resource. The cluster manager uses NGINX Ingress Controller version 0.26.
By default, the cluster manager provides an ingress for Kubernetes Dashboard during the `cm-kubernetes-setup` run, so that the Dashboard works. Port 30080 is the default that is set for the HTTP, and port 30443 is the default that is set for HTTPS.

These 2 ports are exposed on every Kubernetes node, both masters and workers.

The Ingress Controller is deployed as a NodePort which means it comes with a default range of possible port values of 30000-32767.

**NVIDIA device plugin for Kubernetes:** An add-on option in the `cm-kubernetes-setup` run. By default it is not selected. To be used, in addition to being selected, it requires that the NVIDIA GPU drivers are first installed (section 7.4 of the Installation Manual) on the head node and regular node container hosts.

This means that, as described in that section, the regular nodes on which the GPUs are located must have the `cuda-driver` and `cuda-dcgm` packages installed, inside the software image.

After booting up the container host regular nodes, the functioning of the DCGM (Data Center GPU Manager) tools should be checked by running `module load cuda-dcgm; dcgmi config --get`.

The plugin add-on can then be installed as a selection option in the `cm-kubernetes-setup` session that is run by the system administrator after the NVIDIA GPU drivers have been installed. The plugin is somewhat beta at the time of writing (March 2019).

If `cuda-dcgm` and `cuda-driver` are installed for GPUs on the head node, then to ensure that the head node GPUs are detected by CMDaemon, it is recommended to restart CMDaemon on the head node with `service cmdaemon restart`.

The plugin then allows the GPUs to be consumed from the containers. Overcommitting GPUs (sharing) is not possible from containers or pods. Multiple GPUs can be requested by the container. Some documentation on its use can be found at [https://github.com/NVIDIA/k8s-device-plugin](https://github.com/NVIDIA/k8s-device-plugin).

The cluster manager provides NVIDIA device plugin version 1.11

**Kubernetes Setup From The Command Line**

The `cm-kubernetes-setup` command line utility has the following usage synopsis:

```
[root@bright92 ~]# cm-kubernetes-setup -h
usage: Kubernetes Setup cm-kubernetes-setup [-c <config_file>]
    [--cluster CLUSTER_NAME]
    [--skip-docker] [--skip-reboot]
    [--skip-image-update]
    [--add-user USERNAME_ADD]
    [--list-users]
    [--get-user GET_USER]
    [--modify-user USERNAME_MODIFY]
    [--remove-user USERNAME_REMOVE]
    [--namespace NAMESPACE]
    [--add-to-namespace]
    [--remove-from-namespace]
    [--role edit,admin,view,cluster-admin]
    [--runas-uid RNAS_UID]
    [--runas-gids RNAS_GIDS]
    [--user-paths USER_PATHS]
    [--allow-all-uids]
    [--operators OPERATORS]
    [--regenerate-certs]
    [--list-operators] [--psp]
    [--apparmor] [--disable-psp]
    [--update-addons] [--remove]
    [--yes-i-really-mean-it]
    [--remove-ceph-storage] [--pull]
```
optional arguments:
--cluster CLUSTER_NAME
Name of the referred Kubernetes cluster
-h, --help
Print this screen

common:
Common arguments

-c <config_file>
Load runtime configuration for plugins from a YAML config file

installing Kubernetes clusters:
Flags for installing or managing Kubernetes clusters

--skip-docker
Skip the Docker installation steps.
--skip-reboot
Skip the reboot steps.
--skip-image-update
Skip the image update steps.

user management:
Flags for adding a new user in a Kubernetes cluster

--add-user USERNAME_ADD
Create a new user in a Kubernetes cluster
--list-users
Get information about configured Kubernetes users
--get-user GET_USER
Get information about configured Kubernetes users
--modify-user USERNAME_MODIFY
Modify user in a Kubernetes cluster
--remove-user USERNAME_REMOVE
Remove existing user from a Kubernetes cluster

--namespace NAMESPACE
Specify namespace for user (--add-user, --get-user, --modify-user) role binding
(Defaults: USERNAME-restricted for --add-user)

--add-to-namespace
Indicate if permissions to manage namespace needs to be granted for a given user
(--modify-user)

--remove-from-namespace
Indicate if permissions to manage namespace needs to be revoked for a given user
(--modify-user)

--role edit,admin,view,cluster-admin
Specify role for the new (--add-user) and existing (--modify-user) role binding
(Defaults: edit). For 'cluster-admin' namespace flag is ignored

--runas-uid RUNAS_UID
UID is allowed to be used in unprivileged pods (--add-user, --modify-user)

--runas-gids RUNAS_GIDS
Comma-separated list of GIDs allowed to be used in unprivileged pods
(Add-user, --modify-user)

--user-paths USER_PATHS
Comma-separated list of paths user is able to mount in pods (--add-user, --modify-user)

--allow-all-uids Allow user to run processes in pods as any user (--add-user, --modify-user)
hostPath volumes will be disabled for such pods

--operators OPERATORS Comma-separated list of operators user has access to (--add-user, --modify-user)

regenerate Kubernetes certificates:
Flag for regenerating the Kubernetes certificates

--regenerate-certs Regenerate certificates for a Kubernetes cluster

list available operators:
Flag to list available Kubernetes operators

--list-operators List available Kubernetes operators

pod security policies:
Flags for setting up Pod Security Policies

--psp Setup PSP
--apparmor Use AppArmor (Default: false)

disable pod security policies:
Flags for removing Pod Security Policies

--disable-psp Remove PSP

update Kubernetes addons:
Flags for updating Kubernetes addons

--update-addons Update Addons

removing Kubernetes clusters:
Flags for removing a Kubernetes cluster

--remove Remove a Kubernetes cluster
--yes-i-really-mean-it Required for additional safety
--remove-ceph-storage Remove Kubernetes osd pool from Ceph cluster

pulling images to the nodes:
Flags for pulling images to the nodes

--pull Pull images to the nodes
--images IMAGES Comma-separated list of images to pull (--pull)
--nodes NODES Comma-separated list of nodes to pull images to (--pull)
--node-selector NODE_SELECTOR Selector (label query) to filter on, supports '=' ', '==', and '!='.(e.g. key1=value1,key2=value2) (--pull)

Docker storage backend specific:

--allow-device-mapper
Allow to select DeviceMapper (DEPRECATED) storage in wizard

advanced:
Various *advanced* configuration options flags.

- \(v\), \(--\text{verbose}--\) Verbose output
\(--\text{no-distro-checks}--\) Disable distribution checks based on ds.json
\(--\text{json}--\) Use json formatting for log lines printed to stdout
\(--\text{output-remote-execution-runner}--\) Format output for CMDaemon
\(--\text{on-error-action}--\) debug,remotedebug,undo,abort
Upon encountering a critical error, instead of asking the user for choice, setup will do selected action.
\(--\text{skip-packages}--\) Skip the any stages which install packages. Requires packages to be already installed.
\(--\text{min-reboot-timeout}--\) <reboot_timeout_seconds>
How long to wait for nodes to finish reboot (default and minimum allowed: 300 seconds).
\(--\text{dev}--\) Enables additional command line arguments

The `cm-kubernetes-setup` utility should be executed on the console.

Dealing with a pre-existing Docker installation: Docker is required for Kubernetes configured by NVIDIA Bright Cluster Manager. The setup wizard checks if Docker has been installed (page 416), and automatically installs Docker, if needed. However, if Docker has already been configured on the same category of nodes on which Kubernetes is to be installed, then the installation stops, because overriding the existing Docker configuration may not be what is wanted. To override the existing Docker configuration, Docker for that category should first be removed with the `cm-docker-setup --remove` command.

Dealing with a pre-existing Etcd cluster: Etcd is required by Kubernetes to store all the key-value states of the Kubernetes cluster. If no Etcd cluster is found, then the setup wizard prompts to deploy an Etcd cluster. If Etcd is already installed, or present from a previous Kubernetes cluster, then the setup wizard prompts on whether to use the existing Etcd cluster.

Dealing with a pre-existing Harbor installation: Since Harbor uses Docker internally, and because Kubernetes customizes Docker networking, it means that nodes where Harbor is running cannot be reused for Kubernetes.

Kubernetes Setup From A Terminal GUI Session
When the Kubernetes installation is carried out using `cm-kubernetes-setup` without any options, a terminal GUI wizard starts up. The administrator can answer several questions in the wizard. Questions that are asked include questions about which of the node categories or which of the individual nodes should be configured to run the Kubernetes services. There are also questions about the service and pod networks parameters, the port numbers that will be configured for the daemons, whether to install specific add-ons, and so on. After the wizard has been completed, a configuration file can be saved that can be used to set up Kubernetes.

The configuration file can be deployed immediately from the wizard, or it can be deployed later by specifying it as an option to `cm-kubernetes-setup`, in the form `-c <file>`. If no deployment has been carried out earlier, then the main operations screen of the wizard shows just two options, Deploy and Exit. If deployment has already been carried out, then the further options that are available are also displayed (figure 9.2):
Example

[root@bright92 ~]# cm-kubernetes-setup

*Terminal GUI session starts up:*

![Kubernetes Setup Terminal GUI Session: Main Operations Screen After A Deployment](image)

The deployment via CLI or via terminal GUI assigns the appropriate roles, and adds the new Kubernetes cluster. The deployment adds health checks to the monitoring configuration, and it generates certificates for the Kubernetes daemons.

It installs a container runtime:

- The container runtime deployed by default is Containerd (figure 9.3).

- Alternatively, the following Docker runtimes can be deployed:
  - The NVIDIA Bright Cluster Manager Docker runtime
  - A third party Docker package can also be used, for example: an existing (non-Bright) Docker already on the cluster.

However, Docker is deprecated since Kubernetes version 1.21, so neither of the Docker runtime options is recommended.

![Kubernetes Setup Terminal GUI Session: Container Runtime Screen](image)

Calico is set as the Kubernetes network plugin by default. Flannel is an option.

The master, worker, and etcd nodes can be assigned to specific nodes or categories.
The network configuration settings for the Kubernetes cluster can be specified. Ports have default assignments, but can be re-assigned as needed. The etcd spool file path can be set.

The following options are also possible:

- a registry mirror from DockerHub can be specified
- the Kubernetes API server can be exposed to the external network
- the internal network used by Kubernetes nodes can be selected

Add-ons that are available are:

- Ingress Controller (Nginx)
- Kubernetes Dashboard
- Kubernetes Metrics Server
- Kubernetes State Metrics
- Nvidia device plugin for Kubernetes

Operator packages (application managers) that can be installed are:

- cm-jupyter-kernel-operator
- cm-kubernetes-postgresql-operator
- cm-kubernetes-spark-operator

The permission manager—an application for role-based access control—can also be configured.

Testing Kubernetes

To test that Kubernetes works, the `cluster-info` command can be run. For example, on the head node, `bright92`:

**Example**

```
[root@bright92 ~]# module load kubernetes
[root@bright92 ~]# kubectl cluster-info
Kubernetes control plane is running at https://localhost:10443
CoreDNS is running at https://localhost:10443/api/v1/namespaces/kube-system/services/kube-dns:dns/proxy
To further debug and diagnose cluster problems, use 'kubectl cluster-info dump'.
```

After `cm-kubernetes-setup` finishes, and the regular nodes have been rebooted, the state of the nodes can be found by running the `get nodes` command:

**Example**

```
[root@bright92 ~]# kubectl get nodes
NAME     STATUS   ROLES     AGE   VERSION
node001   Ready    <none>   25m   v1.21
node002   Ready    <none>   25m   v1.21
node003   Ready    <none>   25m   v1.21
node004   Ready    <none>   25m   v1.21
node005   Ready    <none>   25m   v1.21
node006   Ready    <none>   25m   v1.21
```
A six node cluster should show the following Kubernetes installed add-ons, when using `kubectl` with the `get all -n kube-system` option (some lines truncated):

**Example**

```
[root@bright92 ~]# module load kubernetes
[root@bright92 ~]# kubectl get all -n kube-system
```

<table>
<thead>
<tr>
<th>NAME</th>
<th>READY</th>
<th>STATUS</th>
<th>RESTARTS</th>
<th>AGE</th>
</tr>
</thead>
<tbody>
<tr>
<td>pod/calico-kube-controllers-58497c65d5-skhgw</td>
<td>1/1</td>
<td>Running</td>
<td>0</td>
<td>26m</td>
</tr>
<tr>
<td>pod/calico-node-27xj7</td>
<td>1/1</td>
<td>Running</td>
<td>0</td>
<td>26m</td>
</tr>
<tr>
<td>pod/calico-node-6hm5</td>
<td>0/1</td>
<td>Running</td>
<td>1</td>
<td>26m</td>
</tr>
<tr>
<td>pod/calico-node-987qv</td>
<td>1/1</td>
<td>Running</td>
<td>0</td>
<td>26m</td>
</tr>
<tr>
<td>pod/calico-node-gcbcm</td>
<td>1/1</td>
<td>Running</td>
<td>0</td>
<td>26m</td>
</tr>
<tr>
<td>pod/calico-node-hisrj</td>
<td>1/1</td>
<td>Running</td>
<td>0</td>
<td>26m</td>
</tr>
<tr>
<td>pod/calico-node-q7k4v</td>
<td>1/1</td>
<td>Running</td>
<td>0</td>
<td>26m</td>
</tr>
<tr>
<td>pod/calico-node-qdbq5</td>
<td>0/1</td>
<td>Running</td>
<td>0</td>
<td>26m</td>
</tr>
<tr>
<td>pod/calico-node-x2drj</td>
<td>1/1</td>
<td>Running</td>
<td>0</td>
<td>26m</td>
</tr>
<tr>
<td>pod/coredns-6768db756-819fs</td>
<td>1/1</td>
<td>Running</td>
<td>0</td>
<td>26m</td>
</tr>
<tr>
<td>pod/coredns-6768db756-cs58q</td>
<td>1/1</td>
<td>Running</td>
<td>0</td>
<td>26m</td>
</tr>
<tr>
<td>pod/kube-state-metrics-758cc75d6-75dsn</td>
<td>1/1</td>
<td>Running</td>
<td>0</td>
<td>26m</td>
</tr>
<tr>
<td>pod/metrics-server-7b477ddf7b9-2drkg</td>
<td>1/1</td>
<td>Running</td>
<td>0</td>
<td>26m</td>
</tr>
<tr>
<td>pod/metrics-server-7b477ddf7b9-z6nch</td>
<td>1/1</td>
<td>Running</td>
<td>0</td>
<td>26m</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>NAME</th>
<th>TYPE</th>
<th>CLUSTER-IP</th>
<th>EXTERNAL-IP</th>
<th>PORT(S)</th>
<th>AGE</th>
</tr>
</thead>
<tbody>
<tr>
<td>service/calico-typha</td>
<td>ClusterIP</td>
<td>10.150.121.25</td>
<td>&lt;none&gt;</td>
<td>5473/TCP</td>
<td>26m</td>
</tr>
<tr>
<td>service/kube-dns</td>
<td>ClusterIP</td>
<td>10.150.255.254</td>
<td>&lt;none&gt;</td>
<td>53/UDP, 53/TCP, 9153/TCP</td>
<td>26m</td>
</tr>
<tr>
<td>service/kube-state-metrics</td>
<td>ClusterIP</td>
<td>None</td>
<td>&lt;none&gt;</td>
<td>8080/TCP, 8081/TCP</td>
<td>26m</td>
</tr>
<tr>
<td>service/metrics-server</td>
<td>ClusterIP</td>
<td>10.150.99.149</td>
<td>&lt;none&gt;</td>
<td>443/TCP</td>
<td>26m</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>NAME</th>
<th>DESIRED</th>
<th>CURRENT</th>
<th>READY</th>
<th>UP-TO-DATE</th>
<th>AVAILABLE</th>
<th>NODE SELECTOR</th>
<th>AGE</th>
</tr>
</thead>
<tbody>
<tr>
<td>daemonset.apps/calico-node</td>
<td>8</td>
<td>8</td>
<td>6</td>
<td>8</td>
<td>6</td>
<td>kubernetes.io/os=linux</td>
<td>26m</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>NAME</th>
<th>READY</th>
<th>UP-TO-DATE</th>
<th>AVAILABLE</th>
<th>AGE</th>
</tr>
</thead>
<tbody>
<tr>
<td>deployment.apps/calico-kube-controllers</td>
<td>1/1</td>
<td>1</td>
<td>1</td>
<td>26m</td>
</tr>
<tr>
<td>deployment.apps/calico-typha</td>
<td>0/0</td>
<td>0</td>
<td>0</td>
<td>26m</td>
</tr>
<tr>
<td>deployment.apps/coredns</td>
<td>2/2</td>
<td>2</td>
<td>2</td>
<td>26m</td>
</tr>
<tr>
<td>deployment.apps/kube-state-metrics</td>
<td>1/1</td>
<td>1</td>
<td>1</td>
<td>26m</td>
</tr>
<tr>
<td>deployment.apps/metrics-server</td>
<td>2/2</td>
<td>2</td>
<td>2</td>
<td>26m</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>NAME</th>
<th>DESIRED</th>
<th>CURRENT</th>
<th>READY</th>
<th>AGE</th>
</tr>
</thead>
<tbody>
<tr>
<td>replicaset.apps/calico-kube-controllers</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>26m</td>
</tr>
<tr>
<td>replicaset.apps/calico-typha</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>26m</td>
</tr>
<tr>
<td>replicaset.apps/coredns-6768db756</td>
<td>2</td>
<td>2</td>
<td>2</td>
<td>26m</td>
</tr>
<tr>
<td>replicaset.apps/kube-state-metrics-758cc75d6</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>26m</td>
</tr>
<tr>
<td>replicaset.apps/metrics-server-7b477ddf7b9</td>
<td>2</td>
<td>2</td>
<td>2</td>
<td>26m</td>
</tr>
</tbody>
</table>

The administrator can now configure the cluster to suit the particular site requirements.

### 9.3.3 Using GPUs With Kubernetes: NVIDIA GPUs

**Prerequisites**

The GPUs have to be recognized by the nodes, and have the appropriate drivers (such as *cuda-driver*) installed. Details on how to do this are given in section 7.4 of the *Installation Manual*.

To verify the GPUs are recognized and have drivers in place, the `nvidia-smi` command can be run. The response displayed for a GPU should look similar to the following:

**Example**

```
``
Docker as a container runtime is deprecated since Kubernetes v1.21 (page 436). It is still possible to use Docker with Kubernetes but it is recommended to deploy Containerd for new Kubernetes setups. The setups for NVIDIA GPUs are described next for both Docker and Containerd:

**Existing Containerd Deployment**

If a non-Bright Containerd has already been deployed before Kubernetes is deployed, then cm-kubernetes-setup may replace an existing Containerd configuration file in order to enable NVIDIA GPU integration via a Kubernetes CNI plugin. This is because Containerd is configured by cm-kubernetes-setup, overwriting any previous configuration.

**Existing Docker Deployment**

If a non-Bright Docker has already been deployed before Kubernetes is deployed, then the instructions from section 9.1.7 must be followed first. These are the instructions on making sure that GPUs can be used inside Docker. The NVIDIA container toolkit (cm-nvidia-container-toolkit) has to be present on the nodes.

To verify Docker is working with GPUs, `nvidia-smi` can be run from inside a container:

```
Example

root@node001:~# module load docker
root@node001:~# docker run --runtime=nvidia --rm nvidia/cuda nvidia-smi
```

root@node001:~# module load docker
root@node001:~# docker run --runtime=nvidia --rm nvidia/cuda nvidia-smi

Mon Sep 28 13:13:39 2020

```
+-----------------------------------------------------------------------------+
| NVIDIA-SMI 450.51.06 Driver Version: 450.51.06 CUDA Version: 11.0          |
|---------------------------------------------------------------------------+
| GPU Name Persistence-M| Bus-Id Disp.A | Volatile Uncorr. ECC | Fan Temp Perf Pwr:Usage/Cap| Memory-Usage | GPU-Util Compute M. | |
| 0 Tesla K40c On | 00000000:00:06.0 Off | Off | 23% 28C P8 22W / 235W | 0MiB / 12206MiB | 0% Default | |
|
+-----------------------------------------------------------------------------+
```
New Kubernetes Installation

If Containerd is selected as container runtime, then cm-kubernetes-setup assigns a new role to the Kubernetes worker nodes: generic::containerd.

The role has a Configurations submode, in which the containerd-cri object can be configured. The entry for Filename specifies the path to the cri.toml file, which contains content used by the container runtime interface on the Kubernetes worker nodes that have been assigned the role.

Example

```
[bright92->configurationoverlay[kube-default-worker]->roles]% use generic::containerd

[...]->roles[generic::containerd]]% show

Parameter  Value
---------------------------------
Name        generic::containerd
Type        GenericRole
Add services yes
Provisioning associations <0 internally used>
Services    containerd
Configurations <2 in submode>
Environments <1 in submode>
Exclude list snippets <1 in submode>
Data node    no

[...]->roles[generic::containerd]]% configurations

[...]->roles[generic::containerd]->configurations% use containerd-cri

[...]->roles[generic::containerd]->configurations[containerd-cri]]% show

Parameter  Value
------------------------------------------
Name        containerd-cri
Type        static
Create directory yes
Filename    /cm/local/apps/containerd/var/etc/conf.d/cri.toml
Filemask directory 0644
User name
Group name
Disabled    no
Service action on write RESTART
Service stop on failure yes
Content    645B
Filemask    0644
```

The file with the CRI (Container Runtime Interface) configuration is created in directory:

```
/cm/local/apps/containerd/var/etc/conf.d
```

and included into the main Containerd configuration file:

```
/cm/local/apps/containerd/var/etc/config.toml
```

with the imports statement:
imports = ['/cm/local/apps/containerd/var/etc/conf.d/*.toml']

Whatever the container runtime that is selected, if NVIDIA GPU integration is required then the NVIDIA container toolkit is taken care of by the installer. During the installation of Kubernetes via `cm-kubernetes-setup` (page 435) there is a step where custom add-ons can be selected (figure 9.4):

![Figure 9.4: Kubernetes Setup Terminal GUI Session: Deployment Of Add-ons](image)

In that step, the add-on `Nvidia device plugin for Kubernetes` should be enabled. This ensures that `cm-nvidia-container-toolkit` is installed in the software image.

This is the default approach. It can be verified to work by deploying a pod that requests a GPU, as requested using the `nvidia.com/gpu: 1` inside the yaml file. Alternatively there is a GPU operator that can be deployed via Helm (page 443)

If more GPUs are available on a single host, then only one GPU should be made visible, and recognized inside the pod, when requesting a single GPU, as in the example:

**Example**

```bash
root@cluster:~> cat gpu.yaml
apiVersion: v1
custom: Pod
metadata:
  name: gpu-pod
spec:
  restartPolicy: Never
  containers:
  - name: cuda-container
    image: nvidia/cuda:9.2-runtime
    command: ["nvidia-smi"]
    resources:
      limits:
        nvidia.com/gpu: 1
root@cluster:~> kubectl apply -f gpu.yaml
pod/gpu-pod configured
root@cluster:~> kubectl logs gpu-pod
Mon Sep 28 12:12:46 2020
+-----------------------------------------------------------------------------+
| NVIDIA-SMI 450.51.06 Driver Version: 450.51.06 CUDA Version: 11.0 |
|-------------------------------+----------------------+----------------------+
| GPU Name Persistence-M| Bus-Id Disp.A | Volatile Uncorr. ECC |
| Fan Temp Perf Pwr:Usage/Cap| Memory-Usage | GPU-Util Compute M. |
|-------------------------------+----------------------+----------------------|
| 0 Tesla K40c On | 00000000:00:06.0 Off | 0% Default |
```

```
Existing Kubernetes Installation

The NVIDIA device plugin is deployed as an add-on. If during the installation the add-on was not selected (section 9.3.3 shows a screenshot), then it can be enabled in cmsh or Bright View afterwards.

```
[cluster->kubernetes[default]->appgroups[system]->applications[nvidia]]% set enabled yes
[cluster->kubernetes[default]->appgroups[system]->applications[nvidia]]% commit
```

This add-on deploys a DaemonSet that will run the NVIDIA device plugin for Kubernetes on nodes with the brightcomputing.com/gpu-accelerator label.

NVIDIA Bright Cluster Manager is responsible for setting this GPU label on the appropriate Kubernetes nodes, with parameter values specified for the labelset:

**Example**

```
[cluster->kubernetes[default]->labelsets[nvidia]]% show
Parameter Value
-------------------------------- ------------------------------------------------
Name nvidia
Revision
Labels brightcomputing.com/gpu-accelerator=
Nodes node001
Categories
Overlays
```

These labels should appear in Kubernetes Node resources as a result, and the device plugin running on these particular nodes (some columns of output have been removed for clarity):

**Example**

```
[root@head ~]# kubectl get nodes --show-labels
NAME       STATUS    ROLES    AGE       VERSION     LABELS
node001    Ready     <none>   16h       v1.21.4     beta.kube...com/gpu-accelerator=...hostname=node001...
node002    Ready     <none>   16h       v1.21.4     beta.kube...com/node-category=de...hostname=node002...
node003    Ready     <none>   16h       v1.21.4     beta.kube...com/node-category=de...hostname=node003...
node004    Ready     <none>   16h       v1.21.4     beta.kube...com/node-category=de...hostname=node004...
node005    Ready     <none>   16h       v1.21.4     beta.kube...com/node-category=de...hostname=node005...
node006    Ready     <none>   16h       v1.21.4     beta.kube...com/node-category=de...hostname=node006...
node007    Ready     <none>   16h       v1.21.4     beta.kube...com/node-category=de...hostname=node007...
head       Ready     master   16h       v1.21.4     beta.kube...com/ingress-controll...hostname=head...

[root@head ~]# kubectl get nodes -l brightcomputing.com/gpu-accelerator=
NAME       STATUS    AGE       VERSION
node001    Ready     16h       v1.21.4
```

```
[root@head ~]# kubectl get pod -l name=nvidia-device-plugin-ds -n kube-system -o wide
NAME               READY STATUS   NODE   NOMINATED NODE   READINESS GATES
nvidia-device-plugin-daemonset-p9x2p 1/1   Running   node001     <none>
```
The NVIDIA GPU Operator ([https://github.com/NVIDIA/gpu-operator](https://github.com/NVIDIA/gpu-operator)) can be deployed using Helm. It leverages the Kubernetes Operator Framework, and has the benefit that it does not require any labels to be set by the user or by NVIDIA Bright Cluster Manager. GPUs should be detected by the operator, and it knows for which nodes the device plugin should be deployed automatically.

For this case, if it is in use, the NVIDIA device plugin add-on configured by Bright should be disabled.

```
[cluster->kubernetes[default]->appgroups[system]->applications[nvidia]]% set enabled no
[cluster->kubernetes*[default*]->appgroups*[system*]->applications*[nvidia*]]% commit
```

The nodes from the labelset can optionally be removed, since brightcomputing.com/gpu-accelerator= is no longer used for anything. The deployment can then proceed using Helm.

In the case of a Docker runtime, the following instructions follow the guidance at [https://docs.nvidia.com/datacenter/cloud-native/gpu-operator/getting-started.html](https://docs.nvidia.com/datacenter/cloud-native/gpu-operator/getting-started.html) (some columns of display output have been removed for clarity):

**Example**

```bash
root@cluster:~> helm repo add nvidia https://nvidia.github.io/gpu-operator
"nvidia" has been added to your repositories
root@cluster:~> helm repo update
Hang tight while we grab the latest from your chart repositories...
...Successfully got an update from the "nvidia" chart repository
Update Complete. #Happy Helming!#
root@cluster:~> helm install nvidia/gpu-operator --wait --generate-name
NAME: gpu-operator-1638875215
LAST DEPLOYED: Tue Dec 7 12:06:58 2021
NAMESPACE: default
STATUS: deployed
REVISION: 1
TEST SUITE: None
```

In the case of a Containerd runtime, the additional option "-{}-set operator.defaultRuntime=containerd" must be added. The following command (provided as an example) sets up the NVIDIA GPU operator with a Containerd runtime, assuming the GPU drivers and toolkit are already deployed on the host:

**Example**

```bash
root@cluster:~> helm install --wait --generate-name nvidia/gpu-operator --set operator.defaultRuntime=containerd --set driver.enabled=false --set toolkit.enabled=false
NAME: gpu-operator-1638875216
LAST DEPLOYED: Tue Dec 21 11:12:01 2021
NAMESPACE: default
STATUS: deployed
REVISION: 1
TEST SUITE: None
```

Uninstalling can be done through Helm as well:

**Example**

```bash
root@cluster:~> helm delete gpu-operator-1638875215
release "gpu-operator-1638875215" uninstalled
```
If the container runtime is Containerd, and the option "-{}-set toolkit.enabled=true" is provided to the Helm command line, then the operator replaces the Containerd configuration file that is on the hosts with its own configuration.

9.3.4 Using GPUs With Kubernetes: AMD GPUs

Prerequisites

The GPUs have to be recognized by the node. One way to check this from within the cluster manager is to run `sysinfo` for the node:

Example

```
[bright92->device[bright92]]% sysinfo | grep GPU
Number of GPUs 1
GPU Driver Version 4.18.0-193.el8.x86_64
GPU0 Name Radeon Instinct MI25
```

In order to make Kubernetes aware of nodes that have AMD GPUs, the AMD GPU device plugin has to be deployed as a DaemonSet inside Kubernetes. The official GitHub repository that hosts this plugin can be found at:

https://github.com/RadeonOpenCompute/k8s-device-plugin

The device plugin requires Kubernetes v1.16+, which has been around since NVIDIA Bright Cluster Manager version 9.0. With some extra instructions, the plugin can also be made a part of NVIDIA Bright Cluster Manager version 8.2.

The DaemonSet YAML file can be deployed with:

Example

```
kubectl create -f https://raw.githubusercontent.com/RadeonOpenCompute/k8s-device-plugin/v1.16/k8s-ds-amdgpu-dp.yaml
```

Managing The YAML File Through CMDaemon

The plugin can be added by the user via the Kubernetes appgroups as an application. In the session that follows, it is given the arbitrary name `device-plugin`:

Example

```
[root@bright92 ]# wget https://raw.githubusercontent.com/RadeonOpenCompute/k8s-device-plugin/v1.16/k8s-ds-amdgpu-dp.yaml -O /tmp/k8s-ds-amdgpu-dp.yaml
[root@bright92 ]# cmsh
[bright92->kubernetes[default]]% appgroups
[bright92->kubernetes[default]->appgroups]% add amd
[bright92->kubernetes[default]->appgroups[amd]]% applications
[bright92->kubernetes[default]->appgroups[amd]->applications]% add device-plugin
```

The configuration of the plugin can be set to the YAML file, by setting the `config` parameter to take the value of the YAML file path.

Example

```
[bright92->...[amd]->applications*[device-plugin*]]% set config /tmp/k8s-ds-amdgpu-dp.yaml
[bright92->kubernetes*[default*]->appgroups*[amd*]->applications*[device-plugin*]]% show Parameter Value
```

---------------------------------
The YAML file can also be edited within cmsh after it has been set, by running `set config` without a value. There are older releases available, starting from Kubernetes v1.10, if needed. Saving this device-plugin YAML should result in pods being scheduled on all the non-tainted nodes, as seen by listing the pods (some columns elided):

```
[root@bright92 ~]# module load kubernetes/default/
[root@bright92 ~]# kubectl get pod -n kube-system -l name=amdgpu-dp-ds -o wide
```

### Including Head Nodes as part of the DaemonSet:

The cluster manager taints head nodes, so that they do not run non-critical pods. The taint can be removed with the `-o` operator to allow non-critical pods to run:

**Example**

```
kubectl taint nodes bright92 node-role.kubernetes.io/master-
```

However, a more specific exception can be configured in the DaemonSet itself.

Within the YAML file, the following existing `tolerations` definition has to be modified, from:

```
tolerations:
- key: CriticalAddonsOnly
  operator: Exists
```

to:

```
tolerations:
- key: node-role.kubernetes.io/master
  effect: NoSchedule
  operator: Exists
```

The modified toleration tolerates this taint, and therefore has the device plugin run on such tainted nodes.

**Verifying that AMD GPUs are recognized by Kubernetes:** If Kubernetes is aware of the AMD GPUs for a node, then several mentions of `amd.com/gpu` are displayed when running the `kubectl describe node` command for the node. The following session shows output for a node `gpu01`, ellipsized for clarity:

**Example**

```
[root@bright92 ~]# kubectl describe node gpu01
Name:       gpu01
... 
Capacity:   
  amd.com/gpu: 3
```
Running The DaemonSet Only On Specific Nodes

The AMD GPU device plugin, unlike the NVIDIA GPU device plugin Daemonset, is scheduled to run on each Kubernetes host. This means that it runs even if the host has no GPU.

This can be prevented with the following steps:

A LabelSet can be created via `cmsh`, and the nodes or categories that have GPUs are assigned within the labelsets mode:

**Example**

```
[root@bright92 ~]# cmsh
[bright92]# kubernetes
[bright92->kubernetes[default]]% labelsets
[bright92->kubernetes[default]->labelsets]% use nvidia
[bright92->kubernetes[default]->labelsets[nvidia]]% .. #but, we're using AMD GPUs, so let's go back up:
[bright92->kubernetes[default]->labelsets]% add amd
[bright92->kubernetes[default]->labelsets[amd]]% set labels brightcomputing.com/amd-gpu-accelerator=
[bright92->kubernetes[default]->labelsets[amd]]% append categories gpu-nodes
[bright92->kubernetes[default]->labelsets[amd]]% commit
```

This assigns the labels to the nodes with GPUs. This can be verified with:

**Example**

```
kubectl get nodes -l brightcomputing.com/amd-gpu-accelerator=
```

<table>
<thead>
<tr>
<th>NAME</th>
<th>STATUS</th>
<th>ROLES</th>
<th>AGE</th>
<th>VERSION</th>
</tr>
</thead>
<tbody>
<tr>
<td>gpu001</td>
<td>Ready</td>
<td>master</td>
<td>66m</td>
<td>v1.18.8</td>
</tr>
<tr>
<td>gpu002</td>
<td>Ready</td>
<td>master</td>
<td>66m</td>
<td>v1.18.8</td>
</tr>
</tbody>
</table>

The DaemonSet YAML can now be adjusted to only run the device plugin on nodes with this new label. This can be done by adding an affinity block after the tolerations block:

**Example**

```
tolerations:
- key: CriticalAddonsOnly  # toleration may be different, if changes were made to it
  operator: Exists

affinity:
  nodeAffinity:
    requiredDuringSchedulingIgnoredDuringExecution:
      nodeSelectorTerms:
      - matchExpressions:
        - key: 'brightcomputing.com/amd-gpu-accelerator'
          operator: Exists
```

This results in the device plugin pods being removed immediately from all nodes that do not have the label.
Running An Example Workload

An example workload can be run as described in the official AMD GPU Kubernetes device plugin documentation at:

https://github.com/RadeonOpenCompute/k8s-device-plugin/tree/v1.16#example-workload

Thus it should now be possible to run:

[root@bright92 ~]# kubectl create -f https://raw.githubusercontent.com/RadeonOpenCompute/k8s-device-plugin/v1.16/example/pod/alexnet-gpu.yaml

The YAML requests only one GPU at the bottom of the YAML file:

```yaml
apiVersion: v1
kind: Pod
metadata:
  name: alexnet-tf-gpu-pod
  labels:
    purpose: demo-tf-amdgpu
spec:
  containers:
    - name: alexnet-tf-gpu-container
      image: rocm/tensorflow:latest
      workingDir: /root
      env:
        - name: HIP_VISIBLE_DEVICES
          value: "0" # # 0,1,2,...,n for running on GPU and select the GPUs, -1 for running on CPU
      command:
        - /bin/bash
        - --
      args:
        - python3 benchmarks/scripts/tf_cnn_benchmarks/tf_cnn_benchmarks.py
        - --model=alexnet;
        - trap : TERM INT; sleep infinity & wait"
      resources: # requesting a GPU
        limits:
          amd.com/gpu: 1

Container creation might take a while due to the image size. Once scheduled, it prints out that it found exactly one GPU, and proceeds to run a TensorFlow workload.

Example

[root@bright92 ~]# kubectl logs -f alexnet-tf-gpu-pod
WARNING:tensorflow:From /usr/local/lib/python3.6/dist-packages/tensorflow/python/compat/v2_compat.py:96:
disable_resource_variables (from tensorflow.python.ops.variable_scope) is deprecated and will be removed
in a future version.
Instructions for updating:
non-resource variables are not supported in the long term
2021-01-08 21:03:29.222293: I tensorflow/core/platform/profile_utils/cpu_utils.cc:104
CPU Frequency: 2495445000 Hz
2021-01-08 21:03:29.222398: I tensorflow/compiler/xla/service/service.cc:168
XLA service 0x39f62f0 initialized for platform Host (this does not guarantee that XLA will be used). Devices:
2021-01-08 21:03:29.223754: I tensorflow/compiler/xla/service/service.cc:168
StreamExecutor device (0): Host, Default Version
2021-01-08 21:03:29.223754: I tensorflow/stream_executor/platform/default/dso_loader.cc:48
Successfully opened dynamic library libamdihip64.so
XLA service 0x3a40bb0 initialized for platform ROCM (this does not guarantee that XLA will be used). Devices:
Had more GPUs been requested, more would have been made available to the container.

For comparison, a CPU version of the container is also available. The official instructions can be referred to for these, too.

### 9.3.5 Kubernetes Configuration Overlays

A list of configuration overlays can be seen from within `configurationoverlay` mode:

**Example**

```
[bright92->configurationoverlay] % list
Name (key)      Priority Nodes Categories Roles
----------------- --------- ----------------- -------------...
kube-default-etcd 500 node001..node003 Etcd::Host
kube-default-master 510 node001..node003 Docker::Host, Kube...
kube-default-worker 500 node004..node006 default Docker::Host, Kube...
```

### 9.3.6 Removing A Kubernetes Cluster

A Kubernetes cluster can be removed using `cm-kubernetes-setup` with the `--remove` and `--yes-i-really-mean-it` options. Also, if there more than one cluster present, then the cluster name must be specified using the `--cluster` parameter.

A removal run looks as follows (some output ellipsized):

**Example**

```
[root@bright92 ~] # cm-kubernetes-setup --remove --cluster default --yes-i-really-mean-it
```

Connecting to CMDaemon

Executing 20 stages

```
# Starting execution for 'Kubernetes Setup'
  - kubernetes
  - docker
# Progress: 0
### stage: kubernetes: Get Kube Cluster
## Progress: 5
### stage: kubernetes: Check Kube Cluster Exists
## Progress: 10
#### stage: kubernetes: Find Installed Components
```
9.3 Kubernetes

Using the --remove option removes the Kubernetes cluster configuration from the cluster manager, unassigns Kubernetes-related roles—including the EtdcHost role—and removes Kubernetes health checks. The command does not remove packages that were installed with a cm-kubernetes-setup command before that.

After the disabling procedure has finished, the cluster has no Kubernetes configured and running.

9.3.7 Kubernetes Cluster Configuration Options

Kubernetes allows many Kubernetes clusters to be configured. These are separated sets of hosts with different certificates, users and other global settings.

When carrying out the Kubernetes setup run, a Kubernetes cluster name will be asked, and a new object with the cluster settings is then added into the CMDaemon configuration. The administrator can change the settings of the cluster from within the kubernetes mode of cmsh or within the Kubernetes Clusters options window of Bright View, accessible via the clickpath Containers→Kubernetes Clusters.

The cmsh equivalent looks like:

Example

```
[root@bright92 ~]# cmsh
[bright92]% kubernetes list
Name (key)
------------------
default
[bright92]% kubernetes use default
```
The preceding `kubernetes` mode parameters are described in table 9.6:

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Authorization Mode</td>
<td>Selects how to authorize on the secure port (default: RBAC (Role-Based Access Control) and Node Authorization modes)</td>
</tr>
<tr>
<td>Kube Config</td>
<td>Path to a kubeconfig file, specifying how nodes authenticate to the API server</td>
</tr>
<tr>
<td>Kube Client Config</td>
<td>Path to a kubeconfig file, specifying how kubelets authenticate to the API server</td>
</tr>
</tbody>
</table>

...continues
<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Kube Config Template</td>
<td>Template the cluster manager uses to generate kubeconfig files for services and users.</td>
</tr>
<tr>
<td>CA</td>
<td>Path to PEM-encoded RSA or ECDSA certificate used for the CA</td>
</tr>
<tr>
<td>CA Key</td>
<td>Path to PEM-encoded RSA or ECDSA private key used for the CA</td>
</tr>
<tr>
<td>Kubernetes Certificate</td>
<td>File containing x509 certificate used by the kubelets</td>
</tr>
<tr>
<td>Kubernetes Key</td>
<td>File containing x509 private key used by the Kubernetes kubelets</td>
</tr>
<tr>
<td>Kubernetes Client</td>
<td>File containing x509 certificate used for the kubelets,</td>
</tr>
<tr>
<td>Certificate</td>
<td></td>
</tr>
<tr>
<td>Kubernetes Client Key</td>
<td>File containing x509 private key used for the Kubernetes kubelets</td>
</tr>
<tr>
<td>Service Accounts</td>
<td>File containing x509 certificate used for Kubernetes service accounts.</td>
</tr>
<tr>
<td>Certificate</td>
<td>This certificate value will be used as --service-account-key-file in the Kubernetes API service.</td>
</tr>
<tr>
<td>Service Accounts Key</td>
<td>File containing x509 private key used for Kubernetes Service Accounts.</td>
</tr>
<tr>
<td></td>
<td>This key value will be used as --service-account-private-key-file in the controller manager service.</td>
</tr>
<tr>
<td>Cluster Domain</td>
<td>Domain for this cluster</td>
</tr>
<tr>
<td>Etcld Cluster</td>
<td>The Etcld cluster instance.</td>
</tr>
<tr>
<td>Etcld Prefix</td>
<td>The prefix for all resource paths in Etcld</td>
</tr>
<tr>
<td>Etcld Servers</td>
<td>List of Etcld servers to watch (format: http://&lt;IP address&gt;[:&lt;port number&gt;])</td>
</tr>
<tr>
<td>Service Network</td>
<td>Network from which the service cluster IP addresses will be assigned, in IPv4 CIDR format. Must not overlap with any IP address ranges assigned to nodes for pods. Default: 172.29.0.0/16</td>
</tr>
<tr>
<td>Pod Network</td>
<td>Network where pod IP addresses will be assigned from</td>
</tr>
<tr>
<td>Internal Network</td>
<td>Network to back the internal communications.</td>
</tr>
</tbody>
</table>
### Parameter Description

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
<th>Option to <code>etcd</code></th>
</tr>
</thead>
<tbody>
<tr>
<td>Trusted Domains</td>
<td>Trusted domains to be included in Kubernetes-related certificates as Alternative Subject Names.</td>
<td></td>
</tr>
<tr>
<td>KubeDNS IP</td>
<td>CoreDNS IP Address.</td>
<td></td>
</tr>
<tr>
<td>Kubernetes API Server</td>
<td>Kubernetes API server address (format: https://&lt;host&gt;:&lt;port number&gt;).</td>
<td></td>
</tr>
<tr>
<td>Proxy Port</td>
<td>Kubernetes API server proxy port.</td>
<td></td>
</tr>
<tr>
<td>Addons</td>
<td>Kubernetes Add-ons managed by CMDaemon.</td>
<td></td>
</tr>
</tbody>
</table>

---

#### 9.3.8 EtcdCluster

The EtcdCluster mode sets the global Etcd cluster settings. It can be accessed via the top level `etcd` mode of `cmsh`.

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
<th>Option to <code>etcd</code></th>
</tr>
</thead>
<tbody>
<tr>
<td>Name</td>
<td>Etc Cluster Name.</td>
<td><code>--initial-cluster-token</code></td>
</tr>
<tr>
<td>Election Timeout</td>
<td>Election Timeout, in milliseconds.</td>
<td><code>--election-timeout</code></td>
</tr>
<tr>
<td>Heart Beat Interval</td>
<td>Heart Beat interval, in milliseconds.</td>
<td><code>--heartbeat-interval</code></td>
</tr>
<tr>
<td>CA</td>
<td>The Certificate Authority (CA) Certificate path for Etcd, used to generate certificates for Etcd.</td>
<td><code>--peer-trusted-ca-file</code></td>
</tr>
<tr>
<td>CA Key</td>
<td>The CA Key path for Etcd, used to generate certificates for Etcd.</td>
<td></td>
</tr>
<tr>
<td>Member Certificate</td>
<td>The Certificate path to use for Etcd cluster members, signed with the Etcd CA. The EtcdHost Role can specify a Member CA as well, and in that case it overwrites any value set here.</td>
<td><code>--peer-cert-file</code></td>
</tr>
<tr>
<td>Member Certificate Key</td>
<td>The Key path to use for Etcd cluster members, signed with the Etcd CA. The EtcdHost Role can specify a Member CA as well, and in that case it overwrites any value set here.</td>
<td><code>--peer-key-file</code></td>
</tr>
</tbody>
</table>

...continues
...continued

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
<th>Option to <code>etcd</code></th>
</tr>
</thead>
<tbody>
<tr>
<td>Client CA</td>
<td>The CA used for client certificates. When set it is assumed client certificate and key will be generated and signed with this CA by another party. Etcd still expects the path to be correct for the Client Certificate and Key.</td>
<td>--trusted-ca-file</td>
</tr>
<tr>
<td>Client Certificate</td>
<td>The Client Certificate, used by Etcdctl for example.</td>
<td>--cert-file</td>
</tr>
<tr>
<td>Key</td>
<td>The Client Certificate Key, used by Etcdctl for example.</td>
<td>--key-file</td>
</tr>
</tbody>
</table>

* Boolean (takes `yes` or `no` as a value)

Table 9.7: EtcdCluster role parameters and etcd options

9.3.9 Kubernetes Roles
Kubernetes roles include the following roles:

- EtcdHost (page 453)
- KubernetesApiServerProxy (page 454)
- KubernetesApiServer (page 454)
- KubernetesController (page 457)
- KubernetesScheduler (page 459)
- KubernetesProxy (page 459)
- KubernetesNode (page 460)

When nodes are configured using Kubernetes roles, then settings in these roles may sometimes use the same pointer variables—for example the Kubernetes or Etcd cluster instance. Pointer variables such as these have definitions that are shared across the roles, as indicated by the parameter description tables for the roles, and which are described in the following pages.

In cmsh, the roles can be assigned:

- for individual nodes via the `roles` submode of `device` mode
- for a category via the `roles` submode of a category
- for a configuration overlay via the `roles` submode of `configurationoverlay` mode

EtcdHost Role
The EtcdHost role is used to configure and manage the `etcd` service for a node.

The `etcd` service manages the `etcd` database, which is a hierarchical distributed key-value database. The database is used by Kubernetes to store its configurations. The EtcdHost role parameters are described in table 9.8:
<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
<th>Option to <code>etcd</code></th>
</tr>
</thead>
<tbody>
<tr>
<td>Member Name</td>
<td>The human-readable name for this <code>etcd</code> member (<code>$hostname</code> will be replaced by the node hostname)</td>
<td><code>--name</code></td>
</tr>
<tr>
<td>Spool</td>
<td>Path to the data directory (default: <code>/var/lib/etcd</code>)</td>
<td><code>--data-dir</code></td>
</tr>
<tr>
<td>Advertise Client URLs</td>
<td>List of client URLs for this member to advertise publicly (default: <code>http://$hostname:5001</code>)</td>
<td><code>--advertise-client-urls</code></td>
</tr>
<tr>
<td>Advertise Peers URLs</td>
<td>List of peer URLs for this member to advertise to the rest of the cluster (default: <code>http://$hostname:5002</code>)</td>
<td><code>--initial-advertise-peer-urls</code></td>
</tr>
<tr>
<td>Listen Peer URLs</td>
<td>List of URLs to listen on for peer traffic (default: <code>http://$hostname:5002</code>)</td>
<td><code>--listen-peer-urls</code></td>
</tr>
<tr>
<td>Snapshot Count</td>
<td>Number of committed transactions that trigger a snapshot to disk (default: 5000)</td>
<td><code>--snapshot-count</code></td>
</tr>
<tr>
<td>Debug</td>
<td>Drop the default log level to DEBUG for all subpackages? (default: no)</td>
<td><code>--debug</code></td>
</tr>
<tr>
<td>Member Certificate</td>
<td>Etcd member certificate, signed with CA specified in the Etcd Cluster. When set it will overrule the value from the EtcdCluster object. Default empty.</td>
<td><code>--peer-cert-file</code></td>
</tr>
<tr>
<td>Member Certificate Key</td>
<td>Etcd member certificate key, signed with CA specified in the Etcd Cluster. When set it will overrule the value from the EtcdCluster object. Default empty.</td>
<td><code>--peer-keyt-file</code></td>
</tr>
<tr>
<td>Options</td>
<td>Additional parameters for the <code>etcd</code> daemon (empty by default)</td>
<td></td>
</tr>
</tbody>
</table>

* Boolean (takes `yes` or `no` as a value)

**Table 9.8:** `EtcdHost` role parameters and `etcd` options

The `etcd` settings are updated by the cluster manager in `/cm/local/apps/etcd/current/etc/cm-etcd.conf`.

**The KubernetesAPI ServerProxy Role**

The KubernetesAPI ServerProxy role sets up a proxy that provides the entry point for one or more instances of the Kubernetes API server. The proxy runs on every node of a Kubernetes cluster instance, including the head node.

If multiple Kubernetes master nodes are present, then it enables HA for the Kubernetes master components, as described in section 9.3.1.

**The KubernetesAPI Server Role**

The KubernetesAPI Server role is used to configure and manage the `kube-apiserver` daemon. The `kube-apiserver` daemon is a Kubernetes API server that validates and configures data for the Kuber-
netes API objects. The API objects include pods, services, and replication controllers. The API Server processes REST operations, and provides a front end to the shared state of the cluster through which all the other components interact.

The KubernetesApiServer role parameters are described in table 9.9:
<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
<th>Option to <code>kube-apiserver</code></th>
</tr>
</thead>
<tbody>
<tr>
<td>Kubernetes Cluster</td>
<td>The Kubernetes cluster instance (pointer)</td>
<td></td>
</tr>
<tr>
<td>Insecure API Port</td>
<td>The port on which to serve unsecured, unauthenticated access (disabled by default)</td>
<td><code>--insecure-port</code></td>
</tr>
<tr>
<td>Secure API Port</td>
<td>The port on which to serve HTTPS with authentication and authorization. If 0, then HTTPS will not be served at all. (default: 6443)</td>
<td><code>--secure-port</code></td>
</tr>
<tr>
<td>Advertise Address</td>
<td>The IP address on which to advertise the API server to members of the cluster with <code>--advertise-address</code>. If set to 0.0.0.0, then the IP address of the management network of the head node is used. (default: 0.0.0.0)</td>
<td></td>
</tr>
<tr>
<td>Insecure Bind Address</td>
<td>IP address to serve on (default: 127.0.0.1)</td>
<td><code>--insecure-bind-address</code></td>
</tr>
<tr>
<td>Secure Bind Address</td>
<td>The IP address on which to serve the read- and secure ports (default: 0.0.0.0)</td>
<td><code>--bind-address</code></td>
</tr>
<tr>
<td>Admission Control</td>
<td>Ordered list of plug-ins to control the admission of resources into the cluster) (default: NamespaceLifecycle,LimitRanger, ServiceAccount,PersistentVolumeLabel, DefaultStorageClass,ValidatingAdmissionWebhook, ResourceQuota,DefaultTolerationSeconds, MutatingAdmissionWebhook)</td>
<td><code>--admission-control</code></td>
</tr>
<tr>
<td>Allowed Privileged*</td>
<td>If yes, then allow privileged containers. (default: no)</td>
<td><code>--allow-privileged</code></td>
</tr>
<tr>
<td>Event TTL</td>
<td>Time period that events are retained. Empty by default. A format example: 1h0m0s</td>
<td><code>--event-ttl</code></td>
</tr>
<tr>
<td>Kubelet Timeout</td>
<td>Kubelet port timeout (default: 5s)</td>
<td><code>--kubelet-timeout</code></td>
</tr>
<tr>
<td>Log Level</td>
<td>Log level (default: 0)</td>
<td><code>--v</code></td>
</tr>
<tr>
<td>Log To StdErr*</td>
<td>Logging to stderr means it goes into the systemd journal (default: yes)</td>
<td><code>--logtostderr</code></td>
</tr>
<tr>
<td>Options</td>
<td>Additional parameters for the <code>kube-apiserver</code> daemon (empty by default)</td>
<td></td>
</tr>
</tbody>
</table>

* Boolean (takes `yes` or `no` as a value)
Further details on the Kubernetes API Server can be found at [https://kubernetes.io/docs/admin/kube-apiserver/](https://kubernetes.io/docs/admin/kube-apiserver/).

**KubernetesController Role**

The Kubernetes Controller role is used to configure and manage the `kube-controller-manager` daemon that embeds the core control loops shipped with Kubernetes. In Kubernetes, a controller is a control loop that watches the shared state of the cluster through the API server, and it makes changes in order to try to move the current state towards the desired state. Examples of controllers that ship with Kubernetes at the time of writing (January 2018) are:

- the replication controller
- the endpoints controller,
- the namespace controller,
- the serviceaccounts controller

The KubernetesController role parameters are described in table 9.10:

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
<th>Option to</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Kubernetes Cluster</strong></td>
<td>The Kubernetes cluster instance (pointer)</td>
<td>--kubernetes-controller-manager</td>
</tr>
<tr>
<td><strong>Address</strong></td>
<td>IP address to serve on (default: 0.0.0.0)</td>
<td>--address</td>
</tr>
<tr>
<td><strong>Port</strong></td>
<td>Port to serve on (default: 10252)</td>
<td>--port</td>
</tr>
<tr>
<td><strong>Concurrent Endpoint Syncs</strong></td>
<td>Number of endpoint syncing operations that will be done concurrently. (default: 5)</td>
<td>--concurrent-endpoint-syncs</td>
</tr>
<tr>
<td><strong>Concurrent Rc Syncs</strong></td>
<td>The number of replication controllers that are allowed to sync concurrently. (default: 5)</td>
<td>--concurrent-rc-syncs</td>
</tr>
<tr>
<td><strong>Namespace Sync Period</strong></td>
<td>Period for syncing namespace life-cycle updates</td>
<td>--namespace-sync-period</td>
</tr>
<tr>
<td><strong>Node Monitor Grace Period</strong></td>
<td>Period for syncing NodeStatus in NodeController</td>
<td>--node-monitor-grace-period</td>
</tr>
<tr>
<td><strong>Node Monitor Period</strong></td>
<td>Period the running Node is allowed to be unresponsive before marking it unhealthy</td>
<td>--node-monitor-period</td>
</tr>
<tr>
<td><strong>Node Startup Grace Period</strong></td>
<td>Period the starting Node is allowed to be unresponsive before marking it unhealthy</td>
<td>--node-startup-grace-period</td>
</tr>
<tr>
<td><strong>Node Sync Period</strong></td>
<td>Period for syncing nodes from cloud-provider</td>
<td>--node-sync-period</td>
</tr>
</tbody>
</table>

...continues
<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
<th>Option to</th>
</tr>
</thead>
<tbody>
<tr>
<td>Pod Eviction</td>
<td>Grace period for deleting pods on failed nodes</td>
<td>--pod-eviction-timeout</td>
</tr>
<tr>
<td>Timeout</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Pvc Claim Binder</td>
<td>Period for syncing persistent volumes and persistent volume claims</td>
<td>--pvcclaimbinder-sync-period</td>
</tr>
<tr>
<td>Sync Period</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Register Retry</td>
<td>Number of retries for initial node registration</td>
<td>--register-retry-count</td>
</tr>
<tr>
<td>Count</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Resource Quota</td>
<td>Period for syncing quota usage status in the system</td>
<td>--resource-quota-sync-period</td>
</tr>
<tr>
<td>Sync Period</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Log Level</td>
<td>Log level (default: 0)</td>
<td>--v</td>
</tr>
<tr>
<td>Log To StdErr*</td>
<td>Logging to stderr means getting it into the systemd journal (default: yes)</td>
<td>--logtostderr</td>
</tr>
<tr>
<td>Options</td>
<td>Additional parameters for the kube-controller-manager daemon</td>
<td></td>
</tr>
<tr>
<td>Cluster signing</td>
<td>Filename containing a PEM-encoded X509 CA certificate used to issue cluster-scoped certificates. (leave empty to use the value of CA defined in the Kubernetes Cluster instance).</td>
<td>--cluster-signing-cert-file</td>
</tr>
<tr>
<td>Cert file</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Cluster signing</td>
<td>Filename containing a PEM-encoded RSA or ECDSA private key used to sign cluster-scoped certificates. (leave empty to use the value of CA Key defined in the Kubernetes Cluster instance).</td>
<td>--cluster-signing-key-file</td>
</tr>
<tr>
<td>Cert Key file</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Use Service Account</td>
<td>Flag to enable or disable use of Service Account Credentials.</td>
<td>--use-service-account-credentials</td>
</tr>
<tr>
<td>Credentials</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Allocate Node Cidrs</td>
<td>Allocate node CIDR in cluster using Pod Network and Node Mask size defined in Kubernetes Cluster Object.</td>
<td>--allocate-node-cidrs</td>
</tr>
<tr>
<td>Kube Config</td>
<td>Path to a kubeconfig file, specifying how to authenticate to API server.</td>
<td>--kubeconfig</td>
</tr>
<tr>
<td>Kubernetes Certificate</td>
<td>File containing x509 Certificate used by Kubernetes Controller Manager. This will be embedded in the Kube Config file.</td>
<td>(1)</td>
</tr>
<tr>
<td>Kubernetes Key</td>
<td>File containing x509 private key used by Kubernetes Controller Manager. This will be embedded in the Kube Config file.</td>
<td>(2)</td>
</tr>
</tbody>
</table>

* Boolean (takes yes or no as a value)

Table 9.10: KubernetesController role parameters and kube-controller-manager options
Further details on the Kubernetes controller manager can be found at https://kubernetes.io/docs/admin/kube-controller-manager/.

**KubernetesScheduler Role**

The KubernetesScheduler role is used to configure and manage the kube-scheduler daemon. The Kubernetes scheduler defines pod placement, taking into account the individual and collective resource requirements, quality of service requirements, hardware/software/policy constraints, affinity and anti-affinity specifications, data locality, inter-workload interference, deadlines, and so on.

The KubernetesScheduler role parameters are described in table 9.11:

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
<th>Option to kube-scheduler</th>
</tr>
</thead>
<tbody>
<tr>
<td>Kubernetes Cluster</td>
<td>The Kubernetes cluster instance (pointer)</td>
<td></td>
</tr>
<tr>
<td>Address</td>
<td>IP address to serve on (default: 0.0.0.0)</td>
<td>--address</td>
</tr>
<tr>
<td>Scheduler Port</td>
<td>Port to serve on (default: 10253)</td>
<td>--port</td>
</tr>
<tr>
<td>Algorithm Provider</td>
<td>The scheduling algorithm provider to use (default: DefaultProvider)</td>
<td>--algorithm-provider</td>
</tr>
<tr>
<td>Policy Config</td>
<td>File with scheduler policy configuration (default: /cm/local/apps/kubernetes/var/etc/scheduler-policy.json)</td>
<td>--policy-config-file</td>
</tr>
<tr>
<td>Log Level</td>
<td>Log level (default: 0)</td>
<td>--v</td>
</tr>
<tr>
<td>Log To StdErr</td>
<td>Logging to STDERR means getting it into the systemd journal (default: yes)</td>
<td>--logtostderr</td>
</tr>
<tr>
<td>Options</td>
<td>Additional parameters for the kube-scheduler daemon</td>
<td></td>
</tr>
<tr>
<td>Kube Config</td>
<td>Path to a kubeconfig file, specifying how to authenticate to API server.</td>
<td>--kubeconfig</td>
</tr>
<tr>
<td>Kubernetes Certificate</td>
<td>File containing x509 Certificate used by Kubernetes Scheduler. This certificate will be embedded in the Kube Config file.</td>
<td></td>
</tr>
<tr>
<td>Kubernetes Key</td>
<td>File containing x509 private key used by Kubernetes Scheduler. This certificate key will be embedded in the Kube Config file.</td>
<td></td>
</tr>
</tbody>
</table>

* Boolean (takes yes or no as a value)

**Table 9.11: KubernetesScheduler role parameters and kube-scheduler options**

Further details on the Kubernetes scheduler can be found at https://kubernetes.io/docs/admin/kube-scheduler/.

**KubernetesProxy Role**

The KubernetesProxy role is used to configure and manage kube-proxy daemon. The kube-proxy daemon runs on each node, and reflects services as defined in the Kubernetes API. It can do simple TCP and UDP stream-forwarding or round-robin TCP and UDP forwarding across a set of back ends.

The KubernetesProxy role parameters are described in table 9.12:
Parameter | Description | Configuration Parameter Passed To 
--- | --- | ---
Kubernetes Cluster | The Kubernetes cluster instance (pointer) | kube-proxy
Address | IP address to serve on (default: 0.0.0.0) | --address
Proxy Port Range Start | Bottom of range of host ports that may be consumed in order to proxy service traffic (not set by default) | --proxy-port-range
Proxy Port Range End | Top of range of host ports that may be consumed in order to proxy service traffic (not set by default) | --proxy-port-range
Masquerade All | If using the pure iptables proxy, SNAT everything (default: yes) | --masquerade-all
Health Check Address | IP address for the health check server to serve on | --healthz-port
Health Check Port | Port to bind the health check server to serve on (default: 10251, use 0 to disable) | --healthz-port
Oom Score Adj | The oom_score_adj value for the kube-proxy process, in range [-999, 999] (default: -999) | --oom-score-adj
Kube Config | Path to a kubecfg file, specifying how to authenticate to the API server. | --kubeconfig
Kubernetes Certificate | File containing x509 Certificate used by kube-proxy. This certificate is embedded in the Kube Config. | 
Kubernetes Key | File containing x509 private key used by Kubernetes API server. This certificate key is embedded in the Kube Config. | 
Log Level | Log level (default: 0) | --v
Log To StdErr* | Logging to STDERR means it goes in the systemd journal (default: yes) | --logtostderr
Options | Additional parameters for the kube-scheduler daemon | 

* Boolean (takes yes or no as a value)

Table 9.12: KubernetesProxy role parameters and kube-proxy options

Further details on the Kubernetes network proxy can be found at [https://kubernetes.io/docs/admin/kube-proxy/](https://kubernetes.io/docs/admin/kube-proxy/).

**KubernetesNode Role**

The KubernetesNode role is used to configure and manage the kubelet daemon, which is the primary node agent that runs on each node. The kubelet daemon takes a set of pod specifications, called PodSpecs, and ensures that the containers described in the PodSpecs are running and healthy.

The KubernetesNode role parameters are described in table 9.13:
<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
<th>Option to <code>kubelet</code></th>
</tr>
</thead>
<tbody>
<tr>
<td>Kubernetes Cluster</td>
<td>The Kubernetes cluster instance (pointer)</td>
<td></td>
</tr>
<tr>
<td>Address</td>
<td>IP address to serve on (default: 0.0.0.0)</td>
<td><code>--address</code></td>
</tr>
<tr>
<td>Kubelet Port</td>
<td>Port that the HTTP service of the node runs on (default: 10250)</td>
<td><code>--port</code></td>
</tr>
<tr>
<td>CNI plugin binaries path</td>
<td>The full path of the directory in which to search for CNI plugin binaries. (default: /cm/local/apps/kubernetes/current/bin/cni)</td>
<td><code>--cni-bin-dir</code></td>
</tr>
<tr>
<td>Enable Server ∗</td>
<td>Enable server mode of Kubelet</td>
<td><code>--enable-server</code></td>
</tr>
<tr>
<td>Host Network Sources</td>
<td>List of sources from which Kubelet allows pods use of the host network (default: file)</td>
<td><code>--host-network-sources</code></td>
</tr>
<tr>
<td>Hostname Override</td>
<td>If non-empty, use this string as identification instead of the actual hostname (not set by default)</td>
<td><code>--hostname-override</code></td>
</tr>
<tr>
<td>Manifests Path</td>
<td>Path to the config file or directory of files (default: /cm/local/apps/kubernetes/var/etc/manifests)</td>
<td><code>--pod-manifest-path</code></td>
</tr>
<tr>
<td>Network plugin</td>
<td>The name of the network plugin to be invoked for various events in kubelet/pod lifecycle. (default: cni)</td>
<td><code>--network-plugin</code></td>
</tr>
<tr>
<td>Spool</td>
<td>Directory path for managing Kubelet files (default: /cm/local/apps/kubernetes/var/kubelet)</td>
<td><code>--root-dir</code></td>
</tr>
<tr>
<td>Cgroup Root</td>
<td>Optional root cgroup to use for pods</td>
<td><code>--cgroup-root</code></td>
</tr>
<tr>
<td>Docker Endpoint</td>
<td>Docker endpoint address to connect to (default: unix:///var/run/docker.sock)</td>
<td><code>--docker-endpoint</code></td>
</tr>
<tr>
<td>Docker Spool</td>
<td>Absolute path to the Docker state root directory (default: /var/lib/docker)</td>
<td><code>--docker-root</code></td>
</tr>
<tr>
<td>Resource Container</td>
<td>Absolute name of the resource-only container to create and run the Kubelet in (default: /kubelet)</td>
<td><code>--resource-container</code></td>
</tr>
<tr>
<td>Allowed Privileged ∗</td>
<td>If true, allow privileged containers. (default: no)</td>
<td><code>--allow-privileged</code></td>
</tr>
<tr>
<td>Labels</td>
<td>List of node labels</td>
<td></td>
</tr>
<tr>
<td>Register On Start ∗</td>
<td>Register the node with the API server (default: yes)</td>
<td><code>--register-node</code></td>
</tr>
<tr>
<td>Eviction minimum reclaim</td>
<td>Minimum amount of resources reclaimed in an eviction (default: imagefs.available=1Gi)</td>
<td><code>--eviction-minimum-reclaim</code></td>
</tr>
</tbody>
</table>

...continues
<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
<th>Option to kubelet</th>
</tr>
</thead>
<tbody>
<tr>
<td>Hard eviction</td>
<td>Hard eviction constraints (default: imagefs.available&lt;1%)</td>
<td>--eviction-hard</td>
</tr>
<tr>
<td>Max Pods</td>
<td>Number of pods that can run on this node</td>
<td>--max-pods</td>
</tr>
<tr>
<td>Max pod eviction grace period</td>
<td>Maximum allowed grace period (in seconds) allowed to terminated pods (default: 60)</td>
<td>--eviction-max-pod-grace-period</td>
</tr>
<tr>
<td>Soft eviction</td>
<td>Soft eviction constraints (default: imagefs.available&lt;5%)</td>
<td>--eviction-soft</td>
</tr>
<tr>
<td>Soft eviction grace period</td>
<td>Soft eviction grace period (default: imagefs.available=1m30s)</td>
<td>--eviction-soft-grace-period</td>
</tr>
<tr>
<td>File Check Frequency</td>
<td>Duration between checking configuration files for new data (default: 20s)</td>
<td>--file-check-frequency</td>
</tr>
<tr>
<td>HTTP Flush Frequency</td>
<td>Duration between checking HTTP for new data (default: 20s)</td>
<td>--http-check-frequency</td>
</tr>
<tr>
<td>Node Status Update Frequency</td>
<td>The absolute free disk space, in MB, to maintain (default: 10s)</td>
<td>--node-status-update-frequency</td>
</tr>
<tr>
<td>Run Once</td>
<td>If true, exit after spawning pods from local manifests or remote URLs (default: no)</td>
<td>--runonce</td>
</tr>
<tr>
<td>Streaming Connection Idle</td>
<td>Maximum time a streaming connection can be idle before the connection is automatically closed (default: 1h)</td>
<td>--streaming-connection-idle-timeout</td>
</tr>
<tr>
<td>Sync Frequency</td>
<td>Maximum period between synchronizing running containers and config (default: 1s)</td>
<td>--sync-frequency</td>
</tr>
<tr>
<td>Image GC High Threshold</td>
<td>Percent of disk usage after which image garbage collection is always run (default: 90)</td>
<td>--image-gc-high-threshold</td>
</tr>
<tr>
<td>Image GC Low Threshold</td>
<td>Percent of disk usage before which image garbage collection is never run (default: 80)</td>
<td>--image-gc-low-threshold</td>
</tr>
<tr>
<td>Oom Score Adjust</td>
<td>The oom_score_adj value for the kubeproxy process, in range [-999, 999] (default: -999)</td>
<td>--oom-score-adj</td>
</tr>
<tr>
<td>Log Level</td>
<td>Log level (default: 0)</td>
<td>--v</td>
</tr>
<tr>
<td>Log To StdErr</td>
<td>Logging to STDERR means it gets into the systemd journal (default: yes)</td>
<td>--logtostderr</td>
</tr>
<tr>
<td>Options</td>
<td>Additional parameters for the kubescraper daemon</td>
<td></td>
</tr>
</tbody>
</table>

...continues
9.3 Kubernetes

Table 9.13: KubernetesNode role parameters and kubelet options

Further details on the kubelet daemon can be found at https://kubernetes.io/docs/admin/kubelet/.

9.3.10 Security Model

The Kubernetes security model allows authentication using a certificate authority (CA), with the user and daemon certificates signed by a Kubernetes CA. The Kubernetes CA should not be confused with the cluster manager CA.

The cluster manager will create a CA specifically for issuing all Kubernetes-related certificates. The certificates are put into /cm/local/apps/kubernetes/var/etc/ by default, and /etc/kubernetes/ is made a link to this directory.

In Kubernetes terminology a user is a unique identity accessing the Kubernetes API server. The user may be a human or an automated process. For example an admin or a developer are human users, but kubelet represents an infrastructure user. Both types of users are authorized and authenticated in the same way against the API server.

Kubernetes uses client certificates, tokens, or HTTP basic authentication methods to authenticate users for API calls. The cluster manager configures client certificate usage by default. The authentication is performed by the API server which validates the user certificate using the common name part of the certificate subject.

In Kubernetes, authorization happens as a separate step from authentication. Authorization applies to all HTTP accesses on the main (secure) API server port. The cluster manager by default enables RBAC (Role-Based Access Control) combined with Node Authorization. The authorization check for any request thus takes the common name and/or organization part of the certificate subject to determine which roles the user or service has associated. Roles carry a certain set of privileges for resources within Kubernetes.

PodSecurityPolicy

The cluster manager also has support for PodSecurityPolicy (PSP) in the Kubernetes API Server. PSP can be explicitly enabled or disabled (section 9.3.14).

For each user, a PSP is generated and assigned using Helm charts generated by Kubernetes Permissions Manager. (section 9.3.15).

The following defaults are applied:

- Users can only mount their own home directory. They cannot mount other paths such as /etc.
- Users cannot run privileged pods.
- Users can only bind on ports higher than 1024.
- Users can run with their own uid and gid.
- It is possible to allow users to run as root, without using hostPath volumes.

There are also

- default role bindings that grant access to the resources covered by the kubectl get all command.

A higher-level explanation about each of the resources is given in section 9.3.13.

These policies only do something if PodSecurityPolicies are enabled, or enforced by Kubernetes, which is not the default. Section 9.3.14 can be referred to for enabling and disabling PodSecurityPolicies.
9.3.11 Addition Of New Kubernetes Users

Bright users can use Kubernetes by making them Kubernetes users. This means having Kubernetes configuration and access set up for them. This can be carried out via the `cm-kubernetes-setup` terminal GUI utility, and choosing the `Add user` option (figure 9.2). The utility then prompts for

- a Kubernetes cluster
- a user name
- a namespace that the privileges are to be assigned to
- a role for the user, with choices provided from:
  - `cluster-admin`: cluster-wide administrator
  - `admin`: administrator
  - `edit`: regular user
  - `view`: read-only user
- a switch if the user is allowed to run as any user, including root, inside pods
- a comma-separated list of paths that the user is able to mount to pods
- the UIDs and GIDs for user processes in pods
- a list of the Kubernetes operators that a user can use

Based on the input, a YAML for the Kubernetes Permission Manager is generated. This in turn, creates a Helm chart with all the required roles, role bindings and pod security policies.

Creation of the user also triggers CMDaemon to create certificate and configuration files in the `.kube` directory.

Adding Users Non-Interactively With `cm-kubernetes-setup`

The `cm-kubernetes-setup` CLI wizard provides the following options:

```
cm-kubernetes-setup -h
usage: Kubernetes Setup cm-kubernetes-setup

[-c <config_file>]
[-cluster CLUSTER_NAME]
[-skip-docker] [-skip-reboot]
[-skip-image-update]
[-add-user USERNAME_ADD] [-list-users] [-get-user GET_USER]
[-modify-user USERNAME_MODIFY] [-remove-user USERNAME_REMOVE]
[-namespace NAMESPACE] [-add-to-namespace] [-remove-from-namespace]
[-role edit,admin,view,cluster-admin]
[-runas-uid RUNAS_UID] [-runas-gids RUNAS_GIDS]
[-user-paths USER_PATHS]
[-allow-all-uids]
[-operators OPERATORS]
[-regenerate-certs]
[-list-operators] [-psp]
[--apparmor] [--disable-psp]
[-update-addons] [-remove]
[-yes-i-really-mean-it]
[-remove-ceph-storage] [-pull]
[-images IMAGES] [-nodes NODES]
[-node-selector NODE_SELECTOR]
[-allow-device-mapper] [-v]
```
The user has to be a user that exists on the cluster already and available via PAM.

If --add-to-namespace is specified, then the namespace has to exist on the Kubernetes cluster already.

Example

```
cm-kubernetes-setup --add-user john
```

The preceding example creates a user john for the default john-restricted namespace. It also assigns the edit role, and gives permission to run processes in the pod with the current UID/GIDs of the user. The ability to mount `/home/john` as a hostPath is also provided.

A way to assign any of the default Kubernetes user-facing roles is also provided by using --role key, as documented at https://kubernetes.io/docs/reference/access-authn-authz/rbac/#user-facing-roles

The possible roles are: view, edit, admin, and cluster-admin.

Example

```
cm-kubernetes-setup --add-user john --role view
```

The preceding example creates a user john with view privileges only, for the default john-restricted namespace.

Example

```
cm-kubernetes-setup --add-user john --namespace development --role edit
```

The preceding example creates a user john, with edit privileges only, for the development namespace. The default john-restricted namespace is not created.

Example

```
cm-kubernetes-setup --add-user john --user-paths /home/john,/scratch --allow-all-uids \
--operators cm-jupyter-kernel-operator
```

The preceding example creates a user john with the following privileges:

- edit privileges
- able to mount `/home/john` and `/scratch` as hostPath volume, when the process runs with UID/GIDs taken from the PAM subsystem on the moment of creation
- able to run as any user, including root (attempts to mount any hostPath volume will be rejected)
- access to the Jupyter Kernel Operator, i.e. with access to the resource kind: `CmKubernetesOperatorPermissionsJupyterKernel`
9.3.12 Getting Information And Modifying Existing Kubernetes Users

It is possible to edit user properties and permissions. `cm-kubernetes-setup` provides 2 ways of doing it: interactively or via CLI options.

Modifying users can be done interactively by choosing Modify User in the `cm-kubernetes-setup` main menu. Guidance is then given on choosing the cluster, users, and on modifying permissions.

Modifying users can also be done via CLI options, by specifying the `--modify-user` argument:

Example

```
cm-kubernetes-setup --add-user john --user-paths /home/john \  
   --allow-all-uids --operators cm-jupyter-kernel-operator
```

In addition to what is specified on creation with the `--add-user` argument in the preceding example, the following example adds permission to mount the `/scratch` hostPath into pods, and gives access to the Zalando PostgreSQL Operator:

```
cm-kubernetes-setup --modify-user john --user-paths /home/john,/scratch \  
   --allow-all-uids --operators cm-jupyter-kernel-operator,cm-kubernetes-postgresql-operator
```

Information about existing users can be found with:

Example

```
cm-kubernetes-setup --list-users
```

Permission for user `john` to operate in the `dev` namespace can be added with:

Example

```
kubectl create namespace dev
```

```
cm-kubernetes-setup --modify-user john --namespace dev --add-to-namespace
```

Permission for the user `john` to operate in the `dev` namespace can be revoked with:

Example

```
kubectl create namespace dev
```

```
cm-kubernetes-setup --modify-user john --namespace dev --remove-from-namespace
```

9.3.13 List Of Resources Defined For Users

These resources are rendered by the Permission Manager Operator, and can therefore be found inside Kubernetes.

The role bindings deployed for every user by default:

- `ClusterRole/john-nodes` (in namespace `john-restricted`)
- `ClusterRoleBinding/john-nodes` (in namespace `john-restricted`)

User `john` is given read-only rights for the Nodes resource (for `kubectl get nodes`).
The secure namespace related resources:  The secure namespace:

- Namespace/john-restricted

The service account used by john:

- ServiceAccount/john (in namespace john-restricted)

This is found referenced, for example, in john’s $HOME/.kube/config.

The PodSecurityPolicy that defines the user can run non-privileged pods, and use only ports above 1024, and so on:

- PodSecurityPolicy/john-restricted (in namespace john-restricted)

More details on this can be found in section 9.3.10, page 463. This policy will only do something as soon as the PodSecurityPolicy Admission Controller is enabled in the API server.

A PodSecurityPolicy that defines the user can run as root as well, but without hostPath volumes:

- PodSecurityPolicy/john-restricted-root (in namespace john-restricted)

To give the aforementioned privileges to john’s secure namespace, so that john can run workloads, execute kubectl get all and more:

- Role/john-restricted (in namespace john-restricted)

- RoleBinding/john-restricted (in namespace john-restricted)

The RoleBinding assigns it to the User john and ServiceAccount account for john. The upstream documentation: https://kubernetes.io/docs/reference/access-authn-authz/service-accounts-admin, has more details on this.

john can be given the ability to use the PodSecurityPolicy defined earlier in his secure namespace, but also in other namespaces:

- ClusterRole/john-psp (in namespace john-restricted)

- ClusterRoleBinding/john-psp (in namespace john-restricted)

The same ability can be given for the second "root but no hostPath" PodSecurityPolicy:

- ClusterRole/john-psp-root (in namespace john-restricted)

- ClusterRoleBinding/john-psp-root (in namespace john-restricted)

Full content of all the documents created for the user can be viewed by checking the generated Helm manifest:

Example

```
helm get manifest -n cm-permissions john-XXXXX
```

9.3.14 Pod Security Policies

Enabling Pod Security Policies For Kubernetes

By default Kubernetes has very few restrictions on users. For a more fine-grained authorization it is possible to enable Pod Security Policies (PSPs).

This can be done via the cm-kubernetes-setup terminal GUI wizard and choosing “Enable PSP”. Alternatively, it can be done non-interactively using cm-kubernetes-setup --psp.

Optionally, AppArmor can be enabled as well, using non-interactive command option --apparmor.

Enabling PSP creates two new Applications within the Kubernetes AppGroup system:
Containerization

• **psp**: this defines the policy and roles for privileged services. The cluster administrator needs to bind these to services typically running in the **kube-system** namespace.

• **psp_system**: this is auto-generated by the wizard. It binds the previously-mentioned privileges to service accounts, for services defined in the system namespaces. This way Calico, CoreDNS, Ingress, and so on, can still function.

The wizard also removes access to the default namespace for existing users, and it restarts the Kubernetes API Server with the **PodSecurityPolicy** feature enabled to enforce all privileges.

Each user should already have their own `<user>-restricted` namespace and privileges to work within this namespace. After the Kubernetes API Server is enabled with the **PodSecurityPolicy** feature, these policies are enforced after the API server has restarted.

A useful command to check exactly what a user—for example, the user **test**—can do is the following:

```
kubectl --kubeconfig=/home/test/.kube/config auth can-i --list --namespace=test-restricted
```

**Disabling Pod Security Policies For Kubernetes**

Disabling PSP can be done via the `cm-kubernetes-setup` terminal GUI wizard, and choosing **Disable PSP**. It can alternatively be carried out non-interactively using `cm-kubernetes-setup --disable-psp`.

When disabling, it should be noted that:

• Existing users are not automatically re-added to the **default** namespace.

• Policies are still defined as resources, but are no longer enforced. This may result in more privileges for users then they had before. That is, they may be able to run as **root** in containers again.

**Enabling Manually Via cmsh Instead**

The PodSecurityPolicy feature is an **admissioncontrol** setting that can be added via cmsh:

```
[cluster->configurationoverlay[kube-default-master]->roles[Kubernetes::ApiServer]]% get admissioncontrol
NamespaceLifecycle
LimitRanger
ServiceAccount
DefaultStorageClass
DefaultTolerationSeconds
MutatingAdmissionWebhook
ValidatingAdmissionWebhook
ResourceQuota
PodSecurityPolicy
```

Adding or removing PodSecurityPolicy from here in cmsh triggers CMDaemon to restart the `kube-apiserver` services.

It could be that running pods re not affected. However, if the cluster administrator re-creates them, then it may be that new pods are not created by ReplicaSets, DaemonSets and similar. Errors may show up as follows:

```
Warning FailedCreate 5m22s (x20 over 5m33s) replicaset-controller Error creating:
```

Also, the `kubelet` services themselves will not have the proper privileges to manage their pods. If the PodSecurityPolicy value is enabled in the admissioncontrol settings, then the cluster administrator must be explicit and define the Pod Security Policy (PSP).
The **psp Application**

The purpose of this application is to define a PSP for these more privileged components. It starts by assigning it to the **kubelet** component. Other system components such as CoreDNS or Calico are not yet assigned to this PSP.

In `cmsh`, parts of the configuration setting can be seen with:

```bash
[cluster->kubernetes[default]->appgroups[system]->applications[psp]]% get config | grep -E "^kind|^ name"
```

```bash
|grep kind -A 2
kind: PodSecurityPolicy
  name: privileged
kind: ClusterRole
  name: privileged-psp
kind: RoleBinding
  name: privileged-psp-nodes
  namespace: kube-system
```

In the preceding configuration, resources define `PodSecurityPolicy/privileged`, which the cluster administrator binds to `Group/system:nodes` with a `RoleBinding/privileged-psp-nodes`, and using a role `ClusterRole/privileged-psp`. The `Group/system:nodes` allows access to resources required by the `kubelet` component, including read access to secrets, and write access to pods.

**The psp application configuration**: The full YAML configuration for the psp application follows:

```yaml
# privileged psp to be used for kube system services only
apiVersion: policy/v1beta1
kind: PodSecurityPolicy
metadata:
  name: privileged
annotations:
  seccomp.security.alpha.kubernetes.io/allowedProfileNames: '*'
spec:
  privileged: true
  allowPrivilegeEscalation: false
  allowedCapabilities: ['*']
  volumes: ['*']
  hostNetwork: true
  hostPorts:
    - min: 0
      max: 65535
  hostIPC: true
  hostPID: true
  runAsUser:
    rule: 'RunAsAny'
  seLinux:
    rule: 'RunAsAny'
  supplementalGroups:
    rule: 'RunAsAny'
  fsGroup:
    rule: 'RunAsAny'
---
# cluster role privileged psp
kind: ClusterRole
apiVersion: rbac.authorization.k8s.io/v1
metadata:
  name: privileged-psp
```

rules:
- apiGroups: ['policy']
  resources: ['podsecuritypolicies']
  verbs: ['use']
  resourceName: ['privileged']
- apiGroups: ['extensions']
  resources: ['podsecuritypolicies']
  verbs: ['use']
  resourceName: ['privileged']

---

# role binding for privileged psp to system:nodes

```yaml
kind: RoleBinding
apiVersion: rbac.authorization.k8s.io/v1
metadata:
  name: privileged-psp-nodes
  namespace: kube-system
roleRef:
  kind: ClusterRole
  name: privileged-psp
  apiGroup: rbac.authorization.k8s.io
subjects:
- kind: Group
  apiGroup: rbac.authorization.k8s.io
  name: system:nodes
```

The `psp_system` Application:
The purpose of this application is to also assign the more privileged PodSecurityPolicy created in the `psp` application to all system applications. To be more specific, this is done by binding it to all the ServiceAccounts and appropriate Namespaces for those services.

In `cmsh` the binding configuration can be viewed with:

```
[cluster->kubernetes[default]->appgroups[system]->applications[psp_system]]% get config | grep -E "^kind|^ name"
| grep kind -A 2
kind: RoleBinding
  name: privileged-psp-calico-kube-controllers
  namespace: kube-system
--
kind: RoleBinding
  name: privileged-psp-nginx-ingress-serviceaccount
  namespace: ingress-nginx
--
kind: RoleBinding
  name: privileged-psp-metrics-server
  namespace: kube-system
--
kind: RoleBinding
  name: privileged-psp-coredns
  namespace: kube-system
--
kind: RoleBinding
  name: privileged-psp-kubernetes-dashboard
  namespace: kubernetes-dashboard
--
kind: RoleBinding
  name: privileged-psp-calico-node
  namespace: kube-system
```
This _psp_system_ is generated by reading all the existing system add-ons from _cmsh_, and binding the _ClusterRole/privileged-psp_ to all _ServiceAccounts_ used by those services with a _RoleBinding/privileged-psp-<serviceaccount>_ for each of them.

**The _psp_system_ application configuration:** The YAML configuration for the _psp_system_ application can be seen with:

```
[cluster->kubernetes[default]->appgroups[system]->applications[psp_system]]% get config
kind: RoleBinding
apiVersion: rbac.authorization.k8s.io/v1
metadata:
  name: privileged-psp-calico-kube-controllers
  namespace: kube-system
roleRef:
  kind: ClusterRole
  name: privileged-psp
  apiGroup: rbac.authorization.k8s.io
subjects:
- kind: ServiceAccount
  name: calico-kube-controllers
  namespace: kube-system
```

The display in the preceding session is truncated, but it is followed by a long list of similar blocks for each of the service accounts used by Calico, DNS, metrics server, device plugins, and so on.

The role has been assigned specifically to ServiceAccounts. One alternative way could be to assign once to the system:serviceaccounts group:

```
kind: RoleBinding
apiVersion: rbac.authorization.k8s.io/v1
metadata:
  name: privileged-psp-system-serviceaccounts
  namespace: kube-system
roleRef:
  kind: ClusterRole
  name: privileged-psp
  apiGroup: rbac.authorization.k8s.io
subjects:
- apiGroup: rbac.authorization.k8s.io
  kind: Group
  name: system:serviceaccounts
```

It should be noted that a few more bindings may still be required for multiple namespaces (kube-system, kube-dashboard, and so on).

**Users And PodSecurityPolicies**
When PodSecurityPolicies have not been enabled from the start, it is possible that users are already running pods that do not necessarily adhere to the new policy. For example pods that have mounted paths outside of their home directory, or are running privileged containers. These may keep working, at least until the Kubernetes scheduler decides to re-schedule them, or until they are terminated.

### 9.3.15 Kubernetes Permission Manager
The Kubernetes permission manager is a custom operator based on Helm. It helps to manage user and system account permissions, roles, role bindings and pod security policies. The operator itself is
packed and distributed as a Helm chart, so it can be installed during Kubernetes cluster creation via the cm-kubernetes-setup terminal GUI. The Helm chart for the operator is located in `/cm/shared/apps/kubernetes-permissions-manager/current/helm`. The output to the following command shows if it is installed:

```
[root@bright92 ~]# module load kubernetes/
[root@bright92 ~]# helm list -n cm
NAME      NAMESPACE  STATUS   ...  CHART                             APP  VERSION
local-path-provisioner cm    deployed ... cm-kubernetes-local-path-provisioner-0.0.20 0.0.20
permissions-manager cm    deployed ... cm-kubernetes-permissions-manager-0.0.1 0.0.1
```

The Helm chart of the operator includes custom resource definitions (CRD), and makes it possible for the administrator to manage resources using the `kubectl` tool:

Example

```
[root@bright92 ~]# cat > permissions.yaml<<EOF
apiVersion: charts.brightcomputing.com/v1alpha1
kind: CmKubernetesPermissionUser
metadata:
  labels:
    namespace: cmsupport-restricted
    username: cmsupport
  name: cmsupport-c7tk7ft
  namespace: cm-permissions
spec:
  allow_all_uids: false
  allowPrivilegeEscalation: false
  allowPrivileged: false
  create_namespace: true
  create_service_account: true
  gids:
    - 1000
  namespace: cmsupport-restricted
  psp_spec_override:
    role: edit
    uid: 1000
  user_paths:
    - /home/cmsupport
  username: cmsupport
EOF
[root@bright92 ~]# kubectl apply -f permissions.yaml
ckubernetespermissionuser.charts.brightcomputing.com/cmsupport-c7tk7ft created
[root@bright92 ~]# kubectl get cmkubernetespermissionusers -A
NAMESPACE  NAME    AGE
cm-permissions cmsupport-c7tk7ft 22s
```

At the time of writing of this section (December 2021), the permission manager handles these 4 CRDs:

1. `cmkubernetespermissionusers` to manage user access to generic resources of the cluster, such as pods, services, secrets, configmaps, etc.
2. `cmkubernetesoperatorpermissionsjupyterkernels` to manage access to the Jupyter Kernels.
3. `cmkubernetesoperatorpermissionspostgresqls` to manage access to the Zalando PostreSQL operator (`https://github.com/zalando/postgres-operator`).

Providing access to third party operators is necessary if pod security policy is enabled. This is because, by default, not only does the user have no access to CRDs, but also the service accounts of the third party operators have no access to the user namespace.

The following example is a YAML document that provides access to the Jupyter Kernel Operator:

```yaml
apiVersion: charts.brightcomputing.com/v1alpha1
kind: CmKubernetesOperatorPermissionsJupyterKernel
metadata:
  labels:
    namespace: cmsupport-restricted
    username: cmsupport
  name: cmsupport-unz4wlf
  namespace: cm-permissions
spec:
  namespace: cmsupport-restricted
  username: cmsupport

Every installed CRD document triggers the Kubernetes permission operator to create a corresponding Helm chart:

```
# helm get values -n cm-permissions cmsupport-unz4wlf
USER-SUPPLIED VALUES:
  namespace: cmsupport-restricted
  username: cmsupport

# helm get manifest -n cm-permissions cmsupport-unz4wlf
---
# Source: cm-kubernetes-operator-permissions-jupyter-kernel/templates/user-permissions.yaml
# Bind policy to service user
apiVersion: rbac.authorization.k8s.io/v1
kind: ClusterRoleBinding
metadata:
  name: cmsupport-unz4wlf-cmsupport-psp
  labels:
    helm.sh/chart: cmsupport-unz4wlf
    app.kubernetes.io/name: cm-kubernetes-operator-permissions-jupyter-kernel
    app.kubernetes.io/instance: cmsupport-unz4wlf
    app.kubernetes.io/version: "0.0.1"
    app.kubernetes.io/managed-by: Helm
subjects:
- kind: ServiceAccount
  name: default
  namespace: cmsupport-restricted
roleRef:
  kind: ClusterRole
  name: cmsupport-psp
  apiGroup: rbac.authorization.k8s.io
```

It is also possible to customize the resulting Helm chart by specifying additional values to specify a section of the CRD. Available values for the Jupyter kernel can be checked using the following command:

```
kubectl exec -it -n cmkpm-system \
  $(kubectl get pods -n cmkpm-system -l control-plane=controller-manager -o name) \
  -it -n cmkpm-system -- /bin/bash
```

---
Similarly, tunables for the generic user permissions of the user are available via:

```
kubectl exec -it -n cmkpm-system \
  $(kubectl get pods -n cmkpm-system -l control-plane=controller-manager -o name) \
  -c manager -- cat /opt/helm/helm-charts/cm-kubernetes-permission-user/values.yaml
```

The output should be similar to:

```
username: "" # name of the user
create_service_account: true # whether to create kubernetes serviceaccount for the user
role: edit # user role
user_paths: [] # hostPath user able to mount to pods
uid: -1 # UID to run process inside pods
gids: [-1] # list of the GIDs for process inside pods
namespace: "" # namespace to give user permissions to
create_namespace: true # create or not the namespace
allowPrivilegeEscalation: false
allowPrivileged: false
allow_all_uids: true # allow or not to run process as any user including root
  # if 'true' and process is run not with user's UID, then
  # all hostPath volumes are denied
psp_spec_override: # custom PSP definition for the user
```

### 9.3.16 Providing Access To External Users

To provide access to users on an external network, the requirements are:

- for `kubectl`, an entry in the company/internal DNS server should resolve the external FQDN to the head node or to one of the nodes where Kubernetes is running;
- for the Kubernetes Dashboard, `dashboard` is a subdomain that must be included as a DNS entry under the external FQDN.

The external FQDN, which is set during the Kubernetes cluster setup, is the first item in the list of trusted domains. This can be retrieved from the Kubernetes cluster entity with `cmsh` as follows:

**Example**

```
[bright92->kubernetes[default]]% get trusteddomains
bright92.example.com
kubernetes
kubernetes.default
kubernetes.default.svc
master
localhost
```

In the preceding example, the FQDN of the cluster is `bright92.example.com`. The cluster administrator managing their own cluster will have another FQDN, and not this FQDN.

For `kubectl`, the Kubernetes API server proxy port should be open to the external network. The proxy port can be retrieved from the Kubernetes cluster entity as follows:

```
[bright92->kubernetes[default]]% get kubernetesapiserverproxyport
10443
```

For the Kubernetes Dashboard, the Ingress Controller HTTPS port should be open to the external network. This port, by default with a value of 30443, can be retrieved from the `ingress_controller` add-on environment:
9.3 Kubernetes

Example

```
[bright92->kubernetes[default]]$ appgroups
[bright92->kubernetes[default]->appgroups]$ applications system
[bright92->kubernetes[default]->appgroups->system]->applications]$ environment ingress_controller
[bright92->...applications[ingress_controller]->environment]$
```

<table>
<thead>
<tr>
<th>Name</th>
<th>Value</th>
<th>Nodes</th>
<th>environment</th>
</tr>
</thead>
<tbody>
<tr>
<td>CM_KUBE_EXTERNAL_FQDN</td>
<td>bright92.example.com</td>
<td>yes</td>
<td></td>
</tr>
<tr>
<td>CM_KUBE_INGRESS_HTTPS_PORT</td>
<td>30443</td>
<td>yes</td>
<td></td>
</tr>
<tr>
<td>CM_KUBE_INGRESS_HTTP_PORT</td>
<td>30080</td>
<td>yes</td>
<td></td>
</tr>
<tr>
<td>ingress_controller_label</td>
<td>brightcomputing.com/ingress-controller</td>
<td>no</td>
<td></td>
</tr>
<tr>
<td>replicas</td>
<td>1</td>
<td>no</td>
<td></td>
</tr>
</tbody>
</table>

If exposing the Kubernetes API server to the external network is selected during setup with `cm-kubernetes-setup`, then the HTTPS and HTTP ports in the preceding example are opened on the Shorewall service that runs on the head node. Exposure to the external network is enabled by default.

**Convention of using a domain name as a prefix label:** In the preceding example, the `brightcomputing.com` prefix that is part of the value for `ingress_controller_label` is just a label rather than a domain. The reason that prefix is used is that it simply follows the convention of using domain names as labels, such as is done by the Kubernetes community (domain: `kubernetes.io`) and RHEL OpenShift (domain: `openshift.io`). The prefix `brightcomputing.com` could equally well have been the prefix `brightaccess` instead. However it is probably less confusing now to follow the established convention. So that is what is done here for the label.

**Users can access the Kubernetes Dashboard:** using `dashboard`. By default, the URL takes the FQDN and the port value along with the `dashboard` subdomain, and has the form:

```
https://dashboard.<CM_KUBEEXTERNAL_FQDN>:<CM_KUBE_INGRESSHTTPS_PORT>
```

So, for example, it could be something like:

Example

```
https://dashboard.bright92.example.com:30443
```

**Ingress configuration for Dashboard in cmsh:** The default Ingress rule described earlier can be found as an object within `cmsh`:

```
[bright92->kubernetes[default]->appgroups[system]->applications[dashboard_ingress]]$ get config
apiVersion: networking.k8s.io/v1beta1
class: Ingress
metadata:
  name: kubernetes-dashboard
  namespace: kubernetes-dashboard
  annotations:
    kubernetes.io/ingress.class: "nginx"
    nginx.ingress.kubernetes.io/secure-backends: "true"
    nginx.ingress.kubernetes.io/ssl-passthrough: "true"
    nginx.ingress.kubernetes.io/backend-protocol: "HTTPS"
spec:
rules:
- host: "dashboard.$CM_KUBE_EXTERNAL_FQDN"
```
Containerization

http:
paths:
  - path: /
    backend:
      serviceName: kubernetes-dashboard
      servicePort: 443

Using kubectl, the Ingress resource can be found with:

```
bash$ kubectl get ingress -n kubernetes-dashboard
NAME HOSTS ADDRESS PORTS AGE
kubernetes-dashboard dashboard.cluster1.local 10.150.153.251 80 45h
```

The official documentation for Ingress, at https://v1.16.docs.kubernetes.io/docs/concepts/services-networking/ingress/, explains it well. Path rewrites without domain names can also be used to set up Ingress with multiple backends (serviceName and servicePort pairs), without having to deal with setting up a DNS.

Ingress Controller running on compute nodes: For scenarios where the head node is not involved in a Kubernetes setup, the cluster manager does not currently set up any forwarding for the Ingress Controller. The cluster manager does set up an NGINX proxy to expose the Kubernetes API Server in such cases, and accessing the Dashboard can then be done with the kubectl proxy approach.

For now a workaround to forward Ingress to a compute node can be achieved with port-forwarding, for example by adding the following line to /etc/shorewall/rules in Shorewall (section 7.2 of the Installation Manual):

Example

```
DNAT net nat:10.141.0.1:30443 tcp 30443
```

Using one Ingress controller for multiple Kubernetes clusters: the cluster manager does not offer an out-of-the-box solution for one Ingress Controller with multiple Kubernetes clusters. This configuration can be achieved by configuring software such as NGINX to proxy, based on the domain name to the appropriate backend(s).

9.3.17 Networking Model

Kubernetes expects all pods to have unique IP addresses, which are reachable from within the cluster. This can be implemented in several ways, including adding pod network interfaces to a network bridge created on each host, or by using 3rd party tools to manage pod virtual networks.

Since NVIDIA Bright Cluster Manager version 9.2, the pod network provider is Calico (https://www.projectcalico.org/). Calico uses the Border Gateway Protocol (BGP) to distribute routes for every Kubernetes pod. This allows the Kubernetes cluster to be integrated without the need for overlays (IP-in-IP). Calico is particularly suitable for large Kubernetes deployments on bare metal, or in private clouds. This is because for larger deployments the performance and complexity costs of overlay networks can become significant.

9.3.18 Kubernetes Monitoring

When cm-kubernetes-setup is run, it configures the following Kubernetes-related health checks:

1. KubernetesChildNode: checks if all the expected agents and services are up and running for active nodes
2. KubernetesComponentsStatus: checks if all the daemons running on a node are healthy
3. KubernetesNodesStatus: checks if Kubernetes nodes have a status of Ready
4. KubernetesPodsStatus: checks if all the pods are in one of these states: Running, Succeeded, or Pending

9.3.19 Local Path Storage Class
For storage, instead of creating Kubernetes PersistentVolumes every time, a modern and practical way is to use the StorageClass feature.

Further documentation on StorageClass is available at:

- https://kubernetes.io/docs/concepts/storage/persistent-volumes/#storageclasses

As a part of initial installation it is possible to choose a Local Path Storage class to utilize the shared storage mounted on every node of the Kubernetes cluster. Possible options include any POSIX shared filesystems, such as NFS, BeeGFS, LustreFS, etc.

![Configure local path storage pool for Kubernetes](image)

Figure 9.5: Kubernetes Terminal GUI Session: Local Storage Configuration

During setup, the installation wizard asks for a path for where Kubernetes physical volumes (PV) will be physically located. This path should be located on a shared filesystem accessible from all nodes.

After installation, the storage class can be seen to be available with:

```
# kubectl get storageclasses.storage.k8s.io
NAME       PROVISIONER RECLAIMPOLICY VOLUMEBINDINGMODE ALLOWVOLUMEEXPANSION AGE
local-path (default) rancher.io/local-path Delete Immediate false 1h
```

Users of the cluster can then freely create physical volume claims (PVC) resources and use them in running pods.

9.3.20 Setup Of A Storage Class For Ceph
Pods running on Kubernetes can use Ceph as a distributed storage system to store data in a persistent way.

This section assumes a working Ceph cluster. Ceph installation for the cluster manager is covered in Chapter 10.

A new pool kube can be created with a replication factor of 3:

**Example**

```
[root@bright92 ~]# ceph osd pool create kube 100 100
pool 'kube' created
[root@bright92 ~]# ceph osd pool set kube size 3
set ppool 1 size to 3
[root@bright92 ~]# ceph osd pool set kube min_size 1
set pool 1 min_size to 1
```

The parameters settings in the preceding example are documented at the Ceph website, at

- http://docs.ceph.com/docs/master/rados/operations/pools/ for documentation on Ceph operations
The pods of a given namespace have to have access to the Ceph RBD images created to back the volumes. A kube client can be created with:

Example

[root@bright92 ~]# ceph auth get-or-create client.kube mon 'allow r' osd 'allow rwx pool=kube'
[client.kube]
key = AQCnOvdZpYewBBAAWv1d7c7/XbEvj7Q07MOTlg==

A list of the current users, and their access control can be viewed with (some output elided):

Example

[root@bright92 ~]# ceph auth list
installed auth entries:
osd.0
  key: AQD9M/dZw8HPNRAAT+x8mG5g6UkjlNqo3GAj4EA==
caps: [mon] allow rwx
caps: [osd] allow *
osd.1
...
client.admin
  key: AQCnM/dZw8HPNRAAT+x8mG5g6UkjlNqo3GAj4EA==
auid: 0
caps: [mds] allow *
caps: [mgr] allow *
caps: [mon] allow *
caps: [osd] allow *
...
client.kube
  key: AQCnOvdZpYewBBAAWv1d7c7/XbEvj7Q07MOTlg==
caps: [mon] allow r
 caps: [osd] allow rwx pool=kube

The admin user must be able to create images in the pool. The admin configuration must therefore look like the section for client.admin in the preceding example.

Similarly, the kube user must be able to map images. The kube configuration must therefore look similar to the section for client.kube in the preceding example.

A Kubernetes secret must be created in the kube-system namespace, using the Ceph admin key:

[root@bright92 ~]# kubectl create secret generic ceph-secret --type="kubernetes.io/rbd" --from-literal=key=$(ceph auth get-key client.admin) --namespace=kube-system
secret "ceph-secret" created

A Kubernetes secret must be created in the default namespace, and in every Kubernetes namespace that needs storage, using the Ceph user key:

[root@bright92 ~]# kubectl create secret generic ceph-secret-user --type="kubernetes.io/rbd" --from-literal=key=$(ceph auth get-key client.kube) --namespace=default
secret "ceph-secret-user" created

Ceph monitor <IP address>:<port> values can be found by running ceph mon stat:
Example

```
[root@bright92 ~]# ceph mon stat
[0] 3 mons at {node001=10.141.0.1:6789/0,node002=10.141.0.2:6789/0,node003=10.141.0.3:6789/0},
  election epoch 38, quorum 0,1,2 node001,node002,node003
```

A storage-class.yml file can then be created, similar to:

```yaml
apiVersion: storage.k8s.io/v1
kind: StorageClass
metadata:
  name: fast
provisioner: kubernetes.io/rbd
parameters:
  monitors: 10.141.0.1:6789,10.141.0.2:6789,10.141.0.3:6789
  adminId: admin
  adminSecretName: ceph-secret
  adminSecretNamespace: kube-system
  pool: kube
  userId: kube
  userSecretName: ceph-secret-user
```

Details about the StorageClass parameters can be found at: https://kubernetes.io/docs/concepts/storage/persistent-volumes/#ceph-rbd

The Kubernetes storage class for Ceph RBD can now be created:

```
[root@bright92 ~]# kubectl apply -f storage-class.yml
storageclass "fast" created
```

To verify it has been created, the new StorageClass can be listed with:

```
[root@bright92 ~]# kubectl get sc
NAME   PROVISIONER
fast   kubernetes.io/rbd
```

### 9.3.21 Integration With Harbor

In order to spawn pods that use images from the Harbor registry, a secret must first be created with the credentials:

```
[root@bright92 ~]# kubectl create secret docker-registry myregistrykey
--docker-server=node001:9443 --docker-username=admin --docker-password=Harbor12345
```

The secret must then be referenced from the pod:

```yaml
apiVersion: v1
kind: Pod
metadata:
  name: foo
spec:
  containers:
    - name: foo
      image: node001:9443/library/nginx
      imagePullSecrets:
        - name: myregistrykey
```

Further information on this is available at https://kubernetes.io/docs/concepts/containers/images/#specifying-imagepullsecrets-on-a-pod.
9.4 Kubernetes Apps

Kubernetes add-ons were introduced in NVIDIA Bright Cluster Manager version 8.1, and could be managed in that version as part of the addons submode of the kubernetes mode in cmsh. In NVIDIA Bright Cluster Manager version 8.2 this feature was expanded into the Kubernetes Applications & Groups feature. Kubernetes Applications & Groups, less formally called app groups, can be accessed via the appgroups submode of cmsh:

**Example**

```
root@bright92 ~# cmsh
[bright92]% kubernetes
[bright92->kubernetes[default]]% appgroups
[bright92->kubernetes[default]->appgroups]% list
Name (key) Applications
------------------- ------------------------------
system              <13 in submode>
```

The version 8.1, addons mode parameters are now accessed from version 8.2 onwards via a default system app group instance. The system instance is accessed in the appgroup submode.

**Example**

```
[bright92->kubernetes[default]->appgroups]% use system
[bright92->kubernetes[default]->appgroups[system]]% show
Parameter                 Value
-------------------------------- ------------------------------------------------
Name                      system
Revision                  
Enabled                   yes
applications              <13 in submode>
```

A Kubernetes application can span multiple namespaces. A name in appgroups therefore only exists to group logically-related applications. Each application contains a YAML configuration file, which the cluster manager synchronizes to the Kubernetes API.

The default system app group is pre-defined. Other app groups can be created as needed. For example, an app group called monitoring could be created to group applications for running Prometheus, node exporters, and anything else related to exposing or viewing Prometheus metrics.

Toggling the Enable parameter of an app group enables or disables all of its application components in Kubernetes. Finer-grained control is possible within the applications mode level, by toggling the
enabled parameter per application component instance. For example, within the calico application component instance:

**Example**

```bash
[bright92->kubernetes[default]->appgroups[system]->applications]% use calico
[bright92->kubernetes[default]->appgroups[system]->applications[calico]]% show
```

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>Name</td>
<td>calico</td>
</tr>
<tr>
<td>Revision</td>
<td></td>
</tr>
<tr>
<td>Format</td>
<td>Yaml</td>
</tr>
<tr>
<td>Enabled</td>
<td>yes</td>
</tr>
<tr>
<td>Config</td>
<td>&lt;25118 bytes&gt;</td>
</tr>
<tr>
<td>Environment</td>
<td>&lt;3 in submode&gt;</td>
</tr>
<tr>
<td>Exclude list</td>
<td>/etc/cni,/opt/cni,/run/calico,/var/lib/calico</td>
</tr>
</tbody>
</table>

A large YAML configuration file for each application component instance can be configured via the Config parameter property, using the `set` option of `cmsh`. This opens up a text editor and allows the environment variables in the YAML configuration file to be managed.

*Exclude list entries* can be set via the `Exclude list` parameter property to prevent the cluster manager software image updates from overwriting files or directories important to the given application.

*Environment entries* can be set via the `Environment` submode. Environment entries are similar to environment variables, and are used to replace variables inside the YAML configuration file. Environment entries can be added to the environment as well, if the `Nodes environment` value inside the `Environment` submode is set to `yes`.

**Example**

```bash
[bright92->kubernetes[default]->appgroups[system]->applications[calico]]% environment
[bright92->kubernetes[default]->appgroups[system]->applications[calico]->environment]% list
```

<table>
<thead>
<tr>
<th>Name</th>
<th>Value</th>
<th>Nodes environment</th>
</tr>
</thead>
<tbody>
<tr>
<td>calico_typha_replicas</td>
<td>0</td>
<td>no</td>
</tr>
<tr>
<td>calico_typha_service</td>
<td>none</td>
<td>no</td>
</tr>
<tr>
<td>head_node_internal_ip</td>
<td>10.141.255.254</td>
<td>no</td>
</tr>
</tbody>
</table>

### 9.4.1 Providing Custom Docker Images

Allowing users to work with custom Docker images on the cluster requires adding the user to the `docker` group. This can be carried out with `usermod -aG docker <user>`.

### 9.5 Kubernetes Operators

Kubernetes operators are the modern way to manage Kubernetes cluster applications ([https://kubernetes.io/docs/concepts/extend-kubernetes/operator/](https://kubernetes.io/docs/concepts/extend-kubernetes/operator/)).

At the time of writing of this section (December 2021), Bright provides and packages several operators which are validated to perform basic functionalities on a Kubernetes Bright setup.

The operators available include:

- the Jupyter Kernel Operator
- the Zalando PostreSQL Operator ([https://github.com/zalando/postgres-operator](https://github.com/zalando/postgres-operator))
Third party operators should be patched to support working in a PSP environment. Not only do users need to have access to the CRDs that these operators bring to the cluster, but the service accounts of the operators also require access to the secure namespaces of the users.

During the initial setup, the installation wizard displays a menu to select which .rpm or .deb packages are to be installed, and to select which Helm charts are to be rolled out into the Kubernetes cluster:

![Figure 9.6: Kubernetes Terminal GUI Session: Selection of operator packages to be installed](image1)

![Figure 9.7: Kubernetes Terminal GUI Session: Selection of operator Helm charts to be installed](image2)

The Helm charts that are selected are installed with sensible defaults. If additional tuning is needed, then the charts can be installed manually after `cm-kubernetes-setup` finishes:

```
yum install cm-kubernetes-postgresql-operator -y
helm install postgres-operator \\
/cm/shared/apps/kubernetes-postgresql-operator/current/helm/postgres-operator-*.tgz
```

If additional tuning is required then tunable values can be set with a command line similar to the following:

```
helm install postgres-operator \\
--values tunables.yaml \\
/cm/shared/apps/kubernetes-postgresql-operator/current/helm/postgres-operator-*.tgz
```

The man page for the operator shows the possible options.

Installed operators can be listed by using the CLI option `--list-operators`:

```
# cm-kubernetes-setup --list-operators
...
#### stage: kubernetes: Display Available Operators
OPERATOR________________________________: api_available___________________________
   cm-jupyter-kernel-operator : true
   cm-kubernetes-postgresql-operator : true
   cm-kubernetes-spark-operator : false
...
```
9.6 Kubernetes On Edge

How edge sites can be configured is described in Chapter 2 of the *Edge Manual*.

If there are Bright Edge sites configured in the cluster, then the Kubernetes setup prompts the user with edge sites that Kubernetes can be deployed on.

![cm-kubernetes-setup prompting for edge sites.](image)

If an edge site is selected, then the rest of the wizard prompts only for nodes available within that edge site; prompts only for the associated network interfaces; and so on.

9.6.1 Flags For Edge Installation

Edge directors often lack high-bandwidth connectivity to the central head node, or they often may benefit from coming up as quickly as possible. It can therefore sometimes be useful to skip stages of the setup.

Running `cm-kubernetes-setup --help` displays some additional flags that allow some setup stages, that bring up a cloud director, to be skipped explicitly:

```bash
cm-kubernetes-setup --help
```

**installing Kubernetes clusters:**

*Flags for installing or managing Kubernetes clusters*

- `--skip-package-install`
  - Skip the package installation steps. Ignores `skip_packages` flags in the config.
- `--skip-reboot`
  - Skip the reboot steps.
- `--skip-image-update`
  - Skip the image update steps.
- `--skip-disksetup-changes`
  - Never change the disk-setup. Use this flag if you manually configure a partition or device for docker thin pool devices for example.

**Speeding Up Kubernetes Installation To Edge Nodes With The `--skip-*` Flags: Use Cases**

Explanations and use cases for these flags are given in the following table:
<table>
<thead>
<tr>
<th>Flag</th>
<th>Use case</th>
</tr>
</thead>
<tbody>
<tr>
<td>(--\text{skip-package-install})</td>
<td>all edge directors share the same software image, and the image is already up to date. So the installer does not need to install packages from that image to the edge director.</td>
</tr>
<tr>
<td>(--\text{skip-image-update}) and (--\text{skip-reboot})</td>
<td>all edge directors are already provisioned with the up-to-date software image. So the installer does not need to carry out an update from the ISO or head node, and then reboot the edge director.</td>
</tr>
<tr>
<td>(--\text{skip-disksetup-changes})</td>
<td>all edge directors already have the correct disk layout. This flag can be set if the disk layout was already configured upfront, in order to avoid full provisioning.</td>
</tr>
</tbody>
</table>

These flags can also be configured in the YAML configuration file of the \texttt{cm-kubernetes-setup} wizard.

The flags can be used for scripted installations for quick Kubernetes setups. For a scripted installation of an edge director, preparations can be done beforehand so that all the requirements in the software images that the edge directors use are already installed, the right disk layouts are already configured, and packages are already updated.

All the stages in the flag options can then be skipped for installing onto edge sites. This can make the setup take just a few seconds per Kubernetes deployment.

### 9.7 Singularity

The cluster manager provides an application containerization tool called Singularity. Singularity is designed to execute containers as if they are just native applications on a host computer, and to work with HPC. Singularity users can therefore run Singularity containers just as they run any other program on an HPC cluster. The cluster manager provides Singularity version 3.8.4.

#### 9.7.1 Use Cases

Adding Singularity to the cluster manager brings a stronger integration of containerization with HPC. While Docker and Kubernetes can work within HPC, some drawbacks still prevent the use of HPC resources in the way that HPC users and administrators are used to.

Besides the use of Singularity containers in HPC jobs, Singularity users can create portable images with their applications. Singularity images are files that represent the container filesystem. These images can be copied from one environment (cluster) to another and executed without modification. Thus, when a user creates a container image file, it is up to the user what files, or which RPMs, to install in the image. For example, the user can create an image file that bundles Open MPI with the user’s application. This guarantees that the application will be able to run if it requires that MPI implementation, even if no MPI libraries are installed on the execution host or if there is some version incompatibility.

There is no need for a special configuration inside workload managers in order to use Singularity. This is because the containers are designed to be run like any application on the system. Users need just to use the image file as the usual script or binary to be executed in their jobscripts or in a shell. The \texttt{singularity} command can also be used to apply special options to the container, when executing the image file in the jobscript or shell.

#### 9.7.2 Package \texttt{cm-singularity}

Singularity is packaged for SLES12, SLES15, Ubuntu 18.04, Ubuntu 20.04, RHEL7/CentOS7, and RHEL8/Rocky Linux 8. It is available from the YUM or Zypper repositories from version 7.3 of NVIDIA Bright Cluster Manager onwards, and is distributed as a package called \texttt{cm-singularity}. The package should be installed in the software image for each node. The user is able to run a Singularity image only if the Singularity package is installed on the node. In order to allow users to build an image, it makes
sense to install the package on the head and login nodes as well. The tool does not provide services that run in the background, so a simple installation of the package is enough to start using it.

Singularity contexts are always run as the user running them. This means that there is no risk in allowing the containers to have access to, and interact with, the file system of the host.

This means that, if an image is created by the root user on a machine, then the files that require root access inside the image, still need to be allowed root permissions on any other machine. Thus, if a user creates an image on a laptop, and adds a file that can be read only by the root user, then when the container is started on another machine by a regular user, that regular user has no access to the root-only readable file inside the container.

While there is no daemon running as root, nor any persistent processes that an attacker may use to escalate privileges, there is a need to run some system calls as root so that the container is encapsulated. For this part of the run flow, there is a single SUID binary called Sexec (Singularity Exec). This is a simple binary that is as small as possible, and which the Singularity developers claim has been audited by multiple security experts.

### 9.7.3 MPI Integration

Because of the nature of Singularity, all MPI implementations should work fine inside a Singularity container. The developers of the tool have spent a lot of effort in making Singularity aware of Open MPI, as well as adding a Singularity module into Open MPI so that running at extreme scale is as efficient as possible. However, in some cases, starting an MPI process may not be as optimal as execution outside the container. So, specifically for Open MPI, Singularity provides a special mechanism to handle the execution of MPI processes. It adds all the MPI processes of the same MPI application to the same container on a host. This also reduces the application startup time. The Open MPI daemon orted in this case is not added to the running container, which means the overhead of starting up daemons is reduced.

When an Open MPI application that has been packaged to an image is started, the following steps take place:

1. `mpirun` is called;
2. `mpirun` forks and executes `orted`;
3. `orted` initializes the PMI (process management interface);
4. `orted` forks as many times as the number of processes per node requested;
5. the container image is started in each fork (because it is the original command specified in `mpirun` arguments);
6. each container process executes the command (that is, the MPI application) passed inside the given container;
7. each of the MPI process links to the dynamic Open MPI library, which loads shared libraries with `dlopen` system call;
8. Open MPI libraries connect back to the original `orted` process via PMI;
9. all non-shared memory communication then occurs through the PMI, and then passes on to local network interfaces.

Additional information about Singularity usage can be found in Chapter 11 of the User Manual. The official web site of the tool is https://www.sylabs.io/singularity.
9.8 OpenShift Container Platform Integration With The Cluster Manager

OpenShift is Red Hat’s container manager. This section describes how OpenShift 4.9 can be installed on a cluster running the cluster manager version 9.2.

The OpenShift cluster deployed in this section is composed of 7 nodes:

- 3 OpenShift master nodes (not managed by Bright)
- 3 RHEL OpenShift compute nodes (managed by Bright and OpenShift)
- 1 RHEL NVIDIA Bright Cluster Manager compute node running a load balancer (managed by Bright).

More OpenShift compute nodes may be added after the initial setup is completed.

The OpenShift master nodes run Red Hat’s Core operating system (RHCOS), and are not Bright-managed. The OpenShift compute nodes are managed by both Bright and OpenShift.

The following official Red Hat documentation can be referred to for further details on OpenShift:

- Installing a cluster on bare metal (OpenShift Container Platform 4.9) (https://docs.openshift.com/container-platform/4.9/installing/installing_bare_metal/installing-bare-metal.html)
- Adding a RHEL compute machine (OpenShift Container Platform 4.9) (https://docs.openshift.com/container-platform/4.9/machine_management/user_infra/adding-rhel-compute.html)

9.8.1 Prerequisites

A cluster with 7 spare compute nodes can be used to install OpenShift 4.9. The minimal hardware specifications for 6 of the OpenShift nodes are:

<table>
<thead>
<tr>
<th>Node Type</th>
<th>Storage</th>
<th>Virtual RAM</th>
<th>vCPU</th>
<th>IOPS</th>
</tr>
</thead>
<tbody>
<tr>
<td>3 masters</td>
<td>100 GB</td>
<td>16 GB</td>
<td>4</td>
<td>300</td>
</tr>
<tr>
<td>1 bootstrap</td>
<td>100 GB</td>
<td>16 GB</td>
<td>4</td>
<td>300</td>
</tr>
<tr>
<td>2 computes</td>
<td>100 GB</td>
<td>8 GB</td>
<td>2</td>
<td>300</td>
</tr>
</tbody>
</table>

Aside from the 6 nodes described above, one extra node is to be used as an L4 (OSI Layer 4, the transport layer) load balancer. Its minimum specifications depend on the expected load of the OpenShift cluster. During the setup process any regular node can handle the load sufficiently.

The head node should have at least 30 GB of spare storage space for the installation.

The supported distribution versions for OpenShift 4.9 use RHEL 7.9 or RHEL 8.4. The Bright cluster is currently limited to running OpenShift on RHEL 7.9.

This session assumes that Kubernetes is not set up.

A prerequisite is that the head node, from where the installation setup is to run, has an active RHEL subscription. A RHEL subscription can be set up by running the following commands:

```
[root@bright92 ~]# subscription-manager register --username=<user_name> --password=<password>
[root@bright92 ~]# subscription-manager refresh
```

9.8.2 Installation

Head Node Setup

The subscription to OpenShift Container Platform should be made. Its details can be searched for by running:

```
[root@bright92 ~]# subscription-manager list --available --matches 'OpenShift Container Platform'
```
One of the outputs in the listing is the pool ID. Using the pool ID that is obtained from the listing, something similar to the following can then be run:

```
[root@bright92 ~]# openshift_version=4.9
[root@bright92 ~]# pool_id=8a85f99b70d0559c0170d55fa6e5c41
[root@bright92 ~]# subscription-manager attach --pool=${pool_id}
[root@bright92 ~]# subscription-manager repos --enable="rhel-7-server-rpms" --enable="rhel-7-server-extras-rpms" --enable="rhel-7-server-ansible-2.8-rpms" --enable="rhel-7-server-ose-${openshift_version}-rpms"
[root@bright92 ~]# yum install -y openshift-ansible openshift-clients jq
```

**Installing OpenShift**

A pull secret file should be picked up from [https://cloud.redhat.com/openshift/install/metal](https://cloud.redhat.com/openshift/install/metal), by clicking on Download pull secret. After placing the file on the head node, the following is run as root on the head node:

```
[root@bright92 ~]# cm-openshift-setup
```

An interactive wizard then walks the cluster administrator through some configuration options for setting up an initial OpenShift cluster.

Some fields in the screens of the wizard are pre-filled with default values.

**Installation Steps**

When running `cm-openshift-setup`, some of the queries concern the following:

- **RHEL Subscription Manager Credentials**: setting these credentials is necessary in order to install the necessary packages into the OpenShift compute nodes.
- **Pull Secret**: The path of the file that was downloaded earlier.
- **Selecting which nodes are to be used as bootstrap, master, compute, and load balancer nodes.**
- **Setting a name for the OpenShift Cluster, and a domain name.** Any name valid for a URL (RFC 1035) works.

There are options which are not asked in the wizard, with default values pre-filled:

- **software_image_name**: The name of the software image used by cmdaemon when provisioning new OpenShift compute nodes.
- **category_name**: The name of the category assigned by cmdaemon to OpenShift compute nodes.
- **openshift_version**: Currently set to 4.9.x. Changing this value is not supported at the time of writing (November 2021).
- **binaries_urls**: The default values can be used initially. In future installations, the administrator may want to host those files at a local server, and provide the path for those files in that server, in order to speed up the process in future installations.
- **ssh_key_path**: The RSA Key used to interact with the nodes. If the key in the path does not yet exist, then a new one is generated, which is recommended. The administrator may provide an existing key instead.
- **openshift_config_dir**: Path to store the OpenShift configurations. The default path should work fine.
- **roles.openshift_load_balancer**: Used to set up the OpenShift load balancer.
- **pxe_template**: Used to populate the PXE boot template, in order to boot RHCOS nodes.
Installation progress can be examined in greater detail in the log file:

**Example**

[root@bright92 ~]# tail -f /var/log/cm-openshift-setup.log

Installation can take around 40 to 90 minutes. After installation is complete, the kubeadmin password can be found in the following path:

[root@bright92 ~]# cat /etc/openshift/auth/kubeadmin-password

### 9.8.3 Adding New Compute Nodes

After the installation is complete, new OpenShift compute nodes can be added into the OpenShift cluster. There are two ways to do so.

1. One way is interactive, using the wizard again:

   [root@bright92 ~]# cm-openshift-setup

   In the main terminal GUI screen of the wizard, the option Add OpenShift Compute Nodes should be selected. The dialog that follows then asks for nodes to be selected with:

   - Select node assigned as the balancer for OpenShift: This is the balancer node that was chosen by the administrator during the installation step
   - Select the nodes to be reprovisioned into OpenShift Compute Nodes: These are the compute nodes.

2. The second way uses a configuration file with the cm-openshift-setup wizard. The configuration file can be generated by cm-openshift-setup after selecting the Add OpenShift Compute Nodes option.

   [root@bright92 ~]# cm-openshift-setup --add-compute-nodes -c <path to cm-openshift-setup config file>

   The nodes that are added are moved to a different category and software image.

### 9.8.4 Validation

All the nodes that were selected during setup can be listed with.

[root@bright92 ~]# oc get nodes

The load balancer does not appear in the list since it is not managed by OpenShift.

A validation check for the cluster operators is:

[root@bright92 ~]# oc get clusteroperators

If all values in the column AVAILABLE are set to True, then the installation has been successful.

### 9.8.5 Uninstall

The following 3 ways of uninstalling OpenShift are possible if it was installed using the cm-openshift-setup wizard.

1. The setup wizard can be run interactively, and the Uninstall OpenShift option can be selected:

   [root@bright92 ~]# cm-openshift-setup

   [terminal GUI screen opens up]
When uninstalling, a confirmation is requested for the values of the fields `unmanaged_config_name`, `category_name` and `software_image_name`. Those fields are pre-filled with the default values. A change is only needed if these were explicitly changed in the installation configuration file generated by the wizard during the installation process (section 9.8.2).

2. The setup wizard can be run non-interactively for an uninstall as follows:

   ```
   [root@bright92 ~]# cm-openshift-setup --remove
   ```

   Run like this, the default values are used.

3. The setup wizard can also be run non-interactively for an uninstall while providing a configuration file for the values, using the following format:

   ```
   [root@bright92 ~]# cm-openshift-setup --remove -c <path to cm-openshift-setup configuration file>
   ```
10 Ceph Installation

10.1 Ceph Introduction

Ceph is a storage software designed for distributed computing. The official Ceph documentation can be found at [http://docs.ceph.com/en/latest/start/intro/](http://docs.ceph.com/en/latest/start/intro/).

NVIDIA Bright Cluster Manager version 9.1 comes with the Octopus (Ceph 15 series) release of Ceph. Octopus has also been backported to NVIDIA Bright Cluster Manager version 9.0.

Ceph Pacific is properly supported for the RHEL8 and Rocky Linux 8 distributions. It is not available for RHEL7.

Nautilus (Ceph 14 series) remains supported for RHEL7.

The current chapter discusses

- The concepts and required hardware for Ceph (section 10.1)
- Ceph installation with `cm-ceph-setup` (section 10.2)
- Ceph installation with Bright View (section 10.3)
- Ceph settings checks, and management with the cluster manager (section 10.4)
- RADOS GW installation and management (section 10.5)

10.1.1 Ceph Object And Block Storage

Ceph is a distributed storage software. It is based on an object store layer called RADOS (Reliable Autonomic Distributed Object Store), which consists of Ceph components called OSDs (Object Storage Daemons) and MONs (Monitoring Servers). These components feature heavily in Ceph. OSDs deal with storing the objects to a device, while MONs deal with mapping the cluster. OSDs and MONs, together carry out object storage and block storage within the object store layer. The Ceph Manager daemon (MGR) runs alongside monitor daemons, to provide additional monitoring and interfaces to external monitoring and management systems. The stack diagram of figure 10.1 illustrates these concepts.
On top of the object store layer are 3 kinds of access layers:

1. **Block device access**: RADOS Block Device (RBD) access can be carried out in two slightly different ways:
   
   (i) via a Linux kernel module based interface to RADOS. The module presents itself as a block device to the machine running that kernel. The machine can then use the RADOS storage, that is typically provided elsewhere.

   (ii) via the *librbd* library, used by virtual machines based on *qemu* or *KVM*. A block device that uses the library on the virtual machine then accesses the RADOS storage, which is typically located elsewhere.

2. **Gateway API access**: RADOS Gateway (RADOS GW) access provides an HTTP REST gateway to RADOS. Applications can talk to RADOS GW to access object storage in a high level manner, instead of talking to RADOS directly at a lower level. The RADOS GW API is compatible with the APIs of Swift and Amazon S3.

3. **Ceph Filesystem access**: CephFS provides a filesystem access layer. A component called MDS (Metadata Server) is used to manage the filesystem with RADOS. MDS is used in addition to the OSD and MON components used by the block and object storage forms when CephFS talks to RADOS.

### 10.1.2 Ceph Storage Backends

OSDs have a choice of two storage backends for managing their data. These are BlueStore and FileStore.

**BlueStore**

BlueStore is a special-purpose storage backend designed specifically for managing data on disk for Ceph OSD workloads. It is the default, and recommended, backend for the Ceph version 13.2.x series onwards.

BlueStore consumes raw block devices or partitions. In contrast to the legacy FileStore approach, BlueStore avoids any intervening layers of abstraction that may limit performance or add complexity.
BlueStore does however, by its design, in contrast to FileStore, require at least an extra volume on the node the OSD runs on. Ceph BlueStore in NVIDIA Bright Cluster Manager versions previous to 8.2 used *ceph-disk* to manage BlueStore devices, while versions 8.2 and beyond use *ceph-volume*, although existing devices managed by *ceph-disk* will continue to work.

**FileStore**
FileStore is the legacy approach to storing objects in Ceph. It relies on a standard file system, which is normally XFS. FileStore is well-tested and widely used in production. However it does suffer from many performance deficiencies due to its overall design and reliance on a traditional file system for storing object data.

Though it is technically possible to store Ceph data alongside other data when using FileStore, it is preferred that dedicated block devices (disks) are used.

The cluster manager supports both storage backends. In the following sections the storage backends are described in some more detail.

### 10.1.3 Ceph Software Considerations Before Use

#### Recommended Filesystem For Legacy FileStore
BlueStore is the recommended storage backend for Ceph. BlueStore requires dedicated block devices (disks) that are fully managed by *ceph-volume* (*ceph-disk* in NVIDIA Bright Cluster Manager prior to version 8.2). The legacy FileStore backend, on the other hand, stores the data directly on a regular file system.

If using FileStore, then recommended file system is XFS, due to its stability, ability to handle extreme storage sizes, and its intrinsic ability to deal with the significant sizes of the extended attributes required by Ceph.

The nodes that run OSDs are typically regular nodes. Within the nodes, the storage devices used by FileStore OSDs automatically have their filesystems configured to be of the XFS type during the installation of Ceph with the cluster manager.

#### Use Of **datanode** For The Protection Of Ceph Data
OSD nodes store the actual data contents of the Ceph cluster. Ceph Monitor nodes also store some data content that is essential for the operation of the Ceph cluster. The devices of these nodes that store such content need protection from being wiped during the reprovisioning that takes place during a reboot of regular nodes.

The recommended way to protect storage devices from being wiped is to set the **datanode** property of their node to **yes** (page 229).

The **datanode** property is automatically set for Monitor and OSD nodes during installation of Ceph with the cluster manager.

#### Use Of Slurm On OSD Nodes
Ceph can be quite demanding of the network and I/O. Running Slurm jobs on an OSD node is therefore not recommended. In addition, if Slurm roles are to be assigned to nodes that have OSD roles, then the default ports 6817 and 6818 used by Slurm can conflict with the default range 6800-7300 used by the Ceph OSD daemons. If there is a need to run Slurm on an OSD node then it is necessary to arrange it so that the ports used do not conflict with each other. During installation, a warning is given when this conflict is present.

### 10.1.4 Hardware For Ceph Use

An **absolute minimum installation**: can be carried out on two nodes, where:

- 1 node, the head node, runs one Ceph Monitor and the first OSD.
- 1 node, the regular node, runs the second OSD.
This is however not recommended, or even supported by the cluster manager. Reasons for why this is not recommended are:

- If the Ceph monitor crashes, and there is no other Ceph monitor running, then Ceph cannot function, and data could be lost.
- The first OSD on the head node requires its own Ceph-compatible filesystem. If that filesystem is not provided, then Ceph on the cluster will run, but in a degraded state.
- Running a monitor service on the same host as an OSD may impair performance due to fsync issues with the kernel.

Using such a system to try to get familiar with how Ceph behaves in a production environment with the cluster manager is unlikely to be worthwhile.

**A more useful minimum:** if there is a node to spare, then it is possible to install Ceph over 3 nodes as follows:

- 1 node, the head node, runs one Ceph Monitor.
- 1 node, the regular node, runs the first OSD.
- 1 more node, also a regular node, runs the second OSD.

In this case the OSD pool default size should be set to 2 in the Global OSD Settings (figure 10.9). Although useful for some testing purposes, this is again not a production system, due to the possible loss of data as well as loss of service if the single Ceph Monitor has issues. This can therefore also not be regarded as a good test cluster.

**For production use:** a redundant number of Ceph Monitor servers is recommended. This is because Ceph Monitors are crucial to Ceph operations. Since the number of Ceph Monitoring servers must be odd, then at least 3 Ceph Monitor servers, with each on a separate node, are recommended for production purposes. The recommended minimum of nodes for production purposes is then 5:

- 2 regular nodes running OSDs.
- 2 regular nodes running Ceph Monitors.
- 1 head node running a Ceph Monitor.

**Drives usable by Ceph:** Ceph OSDs can use any type of disk that presents itself as a block device in Linux. This means that a variety of drives can be used.

### 10.2 Ceph Installation With cm-ceph-setup

Ceph installation for the cluster manager can be carried out with the Ncurses-based cm-ceph-setup utility. It is part of the cm-setup package that comes with the cluster manager. If the Ceph packages are not already installed, then the utility is able to install them for the head and regular nodes, assuming the repositories are accessible, and that the package manager priorities are at their defaults.
10.2.1 Ceph Installation: The Configuration Stage

The cm-ceph-setup utility can be run as root from the head node, and opens up an Ncurses screen (figure 10.2):

Here the administrator may choose to
- Deploy Ceph
- Uninstall Ceph if it is already installed.

**Ceph public network selection:** If the deploy option is chosen, then a screen opens up that allows the selection of the Ceph network used to connect Monitor, OSD and client nodes (figure 10.3):

For a cluster that is configured in a standard default cluster manager Type 1 architecture, the network that is chosen is `internalnet`. In Ceph terminology this is called the **public**, or **front-side**, network. This should not be confused with the informal terminology a cluster manager administrator may sometimes use for a Type 1 architecture, where `externalnet` is sometimes called the public network.

Network architecture types for cluster are discussed in section 3.3.9 of the *Installation Manual*.

**Ceph cluster network selection:** The next screen allows the Ceph **cluster**, or **back-side**, network to be selected (figure 10.4):
The OSDs use this network to rebalance storage.

In a Type 1 architecture this is also typically `internalnet`.

Ceph documentation suggests just using a single public network. In the Type 1 architecture case this is achieved by using the same network, `internalnet`, for the Ceph cluster network as for the Ceph public network.

**Ceph Monitor role assignment to categories:** The OK button in figure 10.3 then brings up a screen that allows the Ceph Monitor role to be assigned to categories (figure 10.5).

**Ceph Monitor role assignment to nodes:** The next screen is similar, and allows Ceph Monitors to be assigned to nodes (figure 10.6):

**Ceph OSD role assignment to categories:** OSD Roles can then be assigned to categories (figure 10.7):
10.2 Ceph Installation With cm-ceph-setup

Ceph OSD role assignment to nodes: If there are any nodes that have not yet been assigned the OSD role, then the next screen (figure 10.8) is similar, and allows the OSD role to be assigned to nodes:

Global Ceph OSD settings: After OSD role assignment is completed, the next screen displayed is the Global Ceph OSD settings screen, (figure 10.9) which allows the OSD pool default size to be set. The OSD pool default size is the default number of replicas for objects in the pool. It should be less than or equal to the number of OSD nodes. If unsure the administrator can just leave it at the default value.

BlueStore device settings: The next screen is the BlueStore configuration screen, which requires that block devices be specified for the OSDs (figure 10.10).

Typically, the administrator would have prepared the nodes that will be taking care of file storage with one or more block devices for BlueStore to use. If these are not there during deployment, then deployment will fail.
BlueStore device settings: The next screen asks if Ceph is to be allowed to remove OSD pools (figure 10.11):

![Figure 10.11: Ceph Installation: Option To Have Ceph Do Removal Of OSD Pools](image)

Summary screen: The summary screen (figure 10.12) allows the configuration to be viewed, saved, or deployed in various combinations.

![Figure 10.12: Ceph Installation: Save Configuration Options](image)

If a save option is chosen, then by default, the configuration is saved to `/root/cm-ceph-setup.conf`

To deploy, the administrator should choose the `Save config & deploy` option:

10.2.2 Ceph Installation: The Deployment Stage

Deployment session output: If deployment is carried out, then the Ncurses screen ends, and session output similar to the following appears:

```
Executing 35 stages
################### Starting execution for 'Ceph Setup'
   - ceph
   ## Progress: 0
### stage: ceph: Networks is available and allowed types.
   Connecting to CMDaemon
   ## Progress: 2
### stage: ceph: Monitor categories and nodes is available.
   ## Progress: 5
### stage: ceph: OSD categories and nodes is available.
   ## Progress: 8
### stage: ceph: Default pool size is correct for selected amount of nodes
   ## Progress: 11
### stage: ceph: Check and normalize device paths to '/dev/<device>' form.
   ## Progress: 14
### stage: ceph: BlueStore configurations is correct and points to devices
   ## Progress: 17
### stage: ceph: FileStore configurations is technically correct.
   ## Progress: 20
### stage: ceph: Collection Nodes Online
   ## Progress: 22
```
### stage: ceph: Get Software Image Paths
## Progress: 25

### stage: ceph: Collection Package Manager Repos Add
## Progress: 28

### stage: ceph: Collection Package Manager Repos Enable
## Progress: 31

### stage: ceph: Collection Packages Installer
## Progress: 37

### stage: ceph: Mark block devices "ClearedOnNextBoot" and "restart_required"
## Progress: 42

### stage: ceph: Collection Nodes Reboot
All nodes to be rebooted: mon002, mon003, osd001, osd002, mon001
Node has been rebooted mon002
Node has been rebooted mon003
Node has been rebooted osd001
Node has been rebooted osd002
Node has been rebooted mon001

Press ctrl+c to abort waiting and continue with deployment

Waiting for nodes to start reboot
Going to wait up to 30 minutes for the nodes to come back up.

Waiting for 5 nodes to come back up
Waiting for 5 nodes to come back up
Waiting for 3 nodes to come back up
All 5 nodes came back up.
## Progress: 45

### stage: ceph: Mark monitors nodes as DataNode
## Progress: 48

### stage: ceph: Mark osd nodes as DataNode
## Progress: 54

### stage: ceph: Create Ceph cluster object
## Progress: 57

### stage: ceph: Bootstrap Monitors
## Progress: 60

### stage: ceph: Assign Monitor Role
Assigning CephMonitorRole role
## Progress: 62

### stage: ceph: Wait Monitors Majority Up
## Progress: 65

### stage: ceph: Assign Ceph Mgr Role
Assigning CephMGRRole role
## Progress: 68

### stage: ceph: Mark OSD nodes with FileStore: restart_required
## Progress: 71

### stage: ceph: Assign Ceph OSD Role
Assigning CephOSDRole role
## Progress: 74

### stage: ceph: Wait OSD Id File Store Assigned
## Progress: 77

### stage: ceph: Configure Prometheus module
Prometheus interface set up correctly
## Progress: 80

### stage: ceph: Load Dashboard module
## Progress: 85

### stage: ceph: Open Shorewall Port On Headnode
Opening port 8443 in Shorewall for ceph dashboard rev. proxy
Restarting shorewall
## Progress: 88
#### stage: ceph: Assign Generic Role
## Progress: 91
#### stage: ceph: Make the Dashboard visible to \bv{}
## Progress: 94
#### stage: ceph: Check Installation
Ceph monitors started
Ceph-manager started
All services started
## Progress: 97
#### stage: ceph: Wait for CEPH OSD is full functional (all OSDs up)
## Progress: 100

Took: 09:21 min.
Progress: 100/100

Finished execution for 'Ceph Setup'. status: completed

Ceph Setup finished!

[root@myhost ~]#

A log of the session is kept at /var/log/cm-ceph-setup.log as well as other relevant logs of installation process.

### 10.3 Installation Of Ceph From Bright View

Ceph can be installed from the cluster manager in the following two ways:

- Using the text-based cm-ceph-setup utility (section 10.2). The utility is a part of the standard cluster-tools package.
- Using the Bright View Ceph Wizard (this section). This is the recommended installation method.

#### 10.3.1 Bright View Ceph Install: Main Details Screen

The clickpath Storage→Ceph→Ceph Wizard brings the browser to the Ceph main details screen, (figure 10.13), if Ceph has not yet been installed by the cluster manager. This screen is beginning of the Ceph installation process, and the page displayed asks for details of the main Ceph configuration settings:
10.3 Installation Of Ceph From Bright View

Figure 10.13: Ceph Wizard Installation: General Cluster Settings

The GUI screen of figure 10.13 is a combination of the Ncurses Ceph Installation General Cluster Settings screen figure 10.3 (page 495), together with the Ncurses OSD journal settings of figure 10.9, (page 497). The settings of the Bright View screen are explained in the texts in the section for figures 10.3 and 10.9.

10.3.2 Bright View Ceph Install: Nodes Selection Screen
The next screen is the Ceph Nodes selection screen (figure 10.14). This allows items to be selected for use as Ceph Monitors and OSDs. The items to be selected can be categories or nodes:
For every selected OSD category or node, the corresponding block devices need to be configured. Clicking on the small settings icon next to the checkbox opens the block device selection dialog (figure 10.15):

Figure 10.14: Ceph Wizard Installation: Ceph Nodes Selection Screen

Figure 10.15: Ceph Wizard Installation: Ceph Block Devices Selection Screen
10.3.3 Bright View Ceph Install: Summary Screen

The next screen is the Summary screen (figure 10.16). This summarizes the choices that have been made. The Show config button displays the underlying raw YAML configuration in a popup window.

![Figure 10.16: Ceph Wizard Installation: Configuration Summary](image)

In figure 10.16 after the Ready for deployment checkbox is checked, the Deploy button proceeds with deploying Ceph according to the configuration specified in the wizard.

10.3.4 Bright View Ceph Install: Deployment Screen

During the deployment process, the progress is displayed (figure 10.17).
The event viewer in Bright View also shows the changes taking place. When deployment is complete, the Finish button ends the wizard.

The state of the deployed system can be checked as shown in section 10.4.1.

### 10.4 Checking And Getting Familiar With Ceph Items After cm-ceph-setup

#### 10.4.1 Checking On Ceph And Ceph-related Files From The Shell

After deployment, the OSD and Monitor services take some time be created and to start up. When all is up and running, the status of a healthy system, according to the output of the ceph -s command, should look something like the following:

**Example**

```
[root@myhost ~]# ceph -s
cluster:
    id:   b4e0fd96-800d-4f66-87f1-79febb102ef5
    health: HEALTH_OK

services:
    mon: 3 daemons, quorum mon001,mon002,mon003
    mgr: mon003(active), standbys: mon001, mon002
    osd: 2 osds: 2 up, 2 in

data:
    pools: 0 pools, 0 pgs
    objects: 0 objects, 0 B
    usage: 2.0 GiB used, 118 GiB / 120 GiB avail
```
The `-h` option to `ceph` lists many options. Users of the cluster manager should usually not need to use these, and should find it more convenient to use the Bright View or `cmsh` front ends instead.

**Generated YAML Configuration File**

A YAML configuration file, by default `cm-ceph-setup.conf`, is generated after a run by the `cm-ceph-setup` utility.

**Using A YAML Configuration File**

The `-c` option to `cm-ceph-setup` allows an existing YAML configuration file to be used.

**Example**

```
[root@bright92 ~]# cm-ceph-setup -c /root/cm-ceph-setup.conf
```

**A Sample YAML Configuration File**

**Example**

```yaml
# This config file should be used with cm-ceph-setup tool
# Example:
# cm-ceph-setup -c <filename>
#
# Generated by:
# cm-ceph-setup
# cluster-tools-8.2-112402_cm8.2.5c7d79cc2f
# cmdline: /cm/local/apps/cm-setup/bin/cm-ceph-setup
# Generate on host:
# bright92
# Date of generation:
# Tue Feb 12 17:43:22 2019
# MD5 checksum of everything after the closing comment:
# 2fc239700ef9f639d69eb7cb9103091a
# to compare: grep -v '^##' <this_file> | md5sum

meta:
  command_line: /cm/local/apps/cm-setup/bin/cm-ceph-setup
  date: Tue Feb 12 17:43:22 2019
  generated_with: Ceph Setup
  hostname: bright92
  package_name: cluster-tools-8.2-112402_cm8.2.5c7d79cc2f
  package_version: '112402'
modules:
  ceph:
    dashboard:
      external_port: 8443
      internal_port: 8444
      password: ceph
      username: ceph
    head_node:
      external_repos:
        - https://openresty.org/package/centos/openresty.repo
    monitors:
      allow_pool_delete: true
      categories: {}
```
nodes:
  mon001: {}
  mon002: {}
  mon003: {}

networks:
  cluster: '
  public: ''

osd:
  FileStore:
    journal_size: 5120
  categories: {}
  nodes:
    osd001:
      BlueStore:
        configurations:
          osd0:
            device: /dev/vdc
      FileStore:
        configurations: {}
        shared_journal:
          device: ''
    osd002:
      BlueStore:
        configurations:
          osd0:
            device: /dev/vdc
      FileStore:
        configurations: {}
        shared_journal:
          device: ''

pool_default_size: 2

packages:
- ceph
- cm-config-ceph-radosgw-systemd
- cm-config-ceph-systemd

prometheus:
  description: Prometheus Ceph plugin
  filter: Prometheus4CephFilter
  name: Prometheus4Ceph
  port: 9283

repos:
- epel
- Ceph
- Ceph-noarch

roles:
  ceph_dashboard_reverse_proxy:
    configurations:
      - content: ceph/templates/nginx.service
        kind: static
        name: service
        path: /usr/lib/systemd/system/ceph-dashboard-reverse-proxy.service
      - kind: template
        name: lua-script
        path: /cm/local/apps/ceph/dashboard/nginx/nginx.lua
        template: ceph/templates/lua.template
- kind: template
  name: config
  path: /cm/local/apps/ceph/dashboard/nginx/nginx.conf
  template: ceph/templates/config.template

env:
  ext_port: 8443
  int_port: 8444

kind: generic
nodes:
  - active
packages:
  - openresty
services:
  - ceph-dashboard-reverse-proxy

For legacy FileStore configurations, the partitioning of Ceph OSD storage devices is done using the
disk setup functionality as described in section 3.9.3. For BlueStore, the corresponding devices are listed
in the CephOSDBlueStoreConfig of the CephOSDRole only, and no entries are added to the XML disk
layout.

Installation Logs
Installation logs to Ceph are kept at:

/var/log/cm-ceph-setup.log

10.4.2 Ceph Management With Bright View And cmsh
Only one instance of Ceph is supported at a time. Its name is ceph.

Ceph Overview And General Properties
From within cmsh, ceph mode can be accessed:

Example

[bright92->ceph]# cmsh
[bright92] ceph
[bright92->ceph]%

From within ceph mode, the overview command lists an overview of Ceph OSDs, MONs, and placement
groups for the ceph instance:

Example

[bright92->ceph]% overview ceph

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>Status</td>
<td>HEALTH_OK</td>
</tr>
<tr>
<td>Number of OSDs</td>
<td>2</td>
</tr>
<tr>
<td>Number of OSDs up</td>
<td>2</td>
</tr>
<tr>
<td>Number of OSDs in</td>
<td>2</td>
</tr>
<tr>
<td>Number of mons</td>
<td>1</td>
</tr>
<tr>
<td>Number of placements groups</td>
<td>192</td>
</tr>
<tr>
<td>Placement groups data size</td>
<td>0B</td>
</tr>
<tr>
<td>Placement groups used size</td>
<td>10.07GB</td>
</tr>
<tr>
<td>Placement groups available size</td>
<td>9.91GB</td>
</tr>
<tr>
<td>Placement groups total size</td>
<td>19.98GB</td>
</tr>
</tbody>
</table>
The Bright View equivalent of the `overview` command is the Ceph Overview window, accessed via the clickpath Storage→Ceph→Ceph Settings→Overview.

Some of the major Ceph configuration parameters can be viewed and their values managed by CM-Daemon from ceph mode. The `show` command shows parameters and their values for the ceph instance:

**Example**

```
[bright92->ceph]% show ceph
```

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>Admin keyring path</td>
<td>/etc/ceph/ceph.client.admin.keyring</td>
</tr>
<tr>
<td>Auto Adjust CRUSH Map</td>
<td>no</td>
</tr>
<tr>
<td>Bootstrapped</td>
<td>yes</td>
</tr>
<tr>
<td>Client admin key</td>
<td>AQCI7klcHwWIBzAAAMUSzd7eYYQiskZELGiaEaA==</td>
</tr>
<tr>
<td>Cluster networks</td>
<td></td>
</tr>
<tr>
<td>Config file path</td>
<td>/etc/ceph/ceph.conf</td>
</tr>
<tr>
<td>Creation time</td>
<td>Thu, 24 Jan 2019 17:57:43 CET</td>
</tr>
<tr>
<td>Extra config parameters</td>
<td>osd journal size = 5120</td>
</tr>
<tr>
<td>Monitor daemon port</td>
<td>6789</td>
</tr>
<tr>
<td>Monitor key</td>
<td>AQCI7klc2bSyAzAAAA5p6p+ljWin75ucxR3gy+Q==</td>
</tr>
<tr>
<td>Monitor keyring path</td>
<td>/etc/ceph/ceph.mon.keyring</td>
</tr>
<tr>
<td>Public networks</td>
<td></td>
</tr>
<tr>
<td>Revision</td>
<td></td>
</tr>
<tr>
<td>auth client required cephx</td>
<td>yes</td>
</tr>
<tr>
<td>auth cluster required cephx</td>
<td>yes</td>
</tr>
<tr>
<td>auth service required cephx</td>
<td>yes</td>
</tr>
<tr>
<td>filestore xattr use omap</td>
<td>no</td>
</tr>
<tr>
<td>fsid</td>
<td>b4e9fd96-800d-4f66-87f1-79febb102ef5</td>
</tr>
<tr>
<td>mon allow pool delete</td>
<td>yes</td>
</tr>
<tr>
<td>mon max osd</td>
<td>10000</td>
</tr>
<tr>
<td>mon osd full ratio</td>
<td>0.950000</td>
</tr>
<tr>
<td>mon osd nearfull ratio</td>
<td>0.850000</td>
</tr>
<tr>
<td>name</td>
<td>ceph</td>
</tr>
<tr>
<td>osd pool default min size</td>
<td>0</td>
</tr>
<tr>
<td>osd pool default pg num</td>
<td>8</td>
</tr>
<tr>
<td>osd pool default pgp num</td>
<td>8</td>
</tr>
<tr>
<td>osd pool default size</td>
<td>2</td>
</tr>
<tr>
<td>rbd cache</td>
<td>yes</td>
</tr>
<tr>
<td>rbd cache max dirty</td>
<td>25165824</td>
</tr>
<tr>
<td>rbd cache max dirty age</td>
<td>1.000000</td>
</tr>
<tr>
<td>rbd cache size</td>
<td>32354432</td>
</tr>
<tr>
<td>rbd cache target dirty</td>
<td>16777216</td>
</tr>
<tr>
<td>rbd cache writethrough until flush</td>
<td>yes</td>
</tr>
<tr>
<td>rbd readahead disable after bytes</td>
<td>52428800</td>
</tr>
<tr>
<td>rbd readahead max bytes</td>
<td>524288</td>
</tr>
<tr>
<td>rbd readahead trigger requests</td>
<td>10</td>
</tr>
<tr>
<td>version</td>
<td>13.2.4</td>
</tr>
</tbody>
</table>
```

The Bright View equivalent of these settings is in the Settings window, accessed via a clickpath of Storage→Ceph→Ceph Settings→Overview→Settings.

**Ceph extraconfigparameters setting:** The Extra config parameters property of a ceph mode object can be used to customize the Ceph configuration file. The Ceph configuration file is typically in `/etc/ceph.conf`, and using extraconfigparameters settings, Ceph can be configured with changes that
CMDaemon would otherwise not manage. After the changes have been set, CMDaemon manages them further.

Thus, the following configuration section in the Ceph configuration file:

```
[mds.2]
host=rabbit
```

could be placed in the file via cmsh with:

**Example**

```
[root@bright92 ~]# cmsh
[bright92]# ceph
[bright92->ceph[ceph]]% append extraconfigparameters "[mds.2] host=rabbit"
[bright92->ceph[ceph]]% commit
```

If a section name, enclosed in square brackets, `[]`, is used, then the section is recognized at the start of an appended line by CMDaemon.

If a section that is specified in the square brackets does not already exist in `/etc/ceph.conf`, then it will be created. The `\n` is interpreted as a new line at its position. After the commit, the extra configuration parameter setting is maintained by the cluster manager.

If the section already exists in `/etc/ceph.conf`, then the associated key=value pair is appended. For example, the following appends `host2=bunny` to an existing `mds.2` section:

```
[bright92->ceph[ceph]]% append extraconfigparameters "[mds.2] host2=bunny"
[bright92->ceph[ceph]]% commit
```

If no section name is used, then the key=value entry is appended to the `[global]` section.

```
[bright92->ceph[ceph]]% append extraconfigparameters "osd journal size = 128"
[bright92->ceph[ceph]]% commit
```

The `/etc/ceph.conf` file has the changes written into it about a minute after the commit, and may then look like (some lines removed for clarity):

```
[global]
auth_client required = cephx
osd journal size=128

[mds.2]
host=rabbit
host2=bunny
```

As usual in cmsh operations (section 2.5.3):

- The `set` command clears `extraconfigparameters` before setting its value
- The `removefrom` command operates as the opposite of the `append` command, by removing key=value pairs from the specified section.

There are similar `extraconfigparameters` for Ceph OSD filesystem associations (page 511) and for Ceph monitoring (page 511).
Ceph OSD Properties

From within ceph mode, the osdinfo command for the Ceph instance displays the nodes that are providing OSDs along with their OSD IDs:

Example

```
[bright92->ceph] % osdinfo ceph
OSD id    Node    OSD name
------------ ---------------------- ------------
0 node001   osd0
1 node002   osd0
```

Within a device or category mode, the roles submode allows parameters of an assigned cephosd role to be configured and managed.

Example

```
[bright92->device[node001]->roles] % show cephosd
Parameter      Value
------------------- -------------------------------
Add services    yes
Name            cephosd
OSD configurations <1 in submode>
Provisioning associations <0 internally used>
Revision
Type            CephOSDRole
```

Within the cephosd role the templates for OSD filesystem configurations, osdconfigurations, can be set or modified:

Example

```
[bright92->device[node001]->roles] % use cephosd
[bright92...[node001]->roles[cephosd]] % osdconfigurations
[bright92...osd] % osdconfigurations % show osd0
Parameter      Value
------------------- -------------------------------
Automatically adjust weight    off
Extra config parameters
Initial weight    0.1
Journal data      /var/lib/ceph/osd/$cluster-$id/journal
Journal size      0 MiB
Name              osd0
OSD data          /var/lib/ceph/osd/$cluster-$id
Production weight 1
Revision
Type              CephOSDLegacyConfig
Weight adjust interval 5
Weight adjust rate 0.1
Weight interpretation scale
```

The Bright View equivalent to access the preceding cmsh OSD configuration settings is via the role for a particular node or category. The clickpath that brings up these configuration options for node node001 is, for example:

```
Devices→Physical Nodes→node001→Edit→Settings→Roles→cephosd→Edit→osd0→Edit
```
OSD filesystem association extraconfigparameters setting: Extra configuration parameters can be set for an OSD filesystem association such as osd0 by setting values for its extraconfigparameters option. This is similar to how it can be done for Ceph general configuration (page 508):

Example

```
[bright92...osd]-->osdconfigurations]% use osd0
[bright92...osdconfigurations[osd0]]% show
Parameter                        Value
-------------------------------- ----------------------------------
... Automatically adjust weight    off
Extra config parameters...
... [bright92...osdconfigurations[osd0]]% set extraconfigparameters "a=b"
...
```

Ceph Monitoring Properties

Similarly to Ceph OSD properties, the parameters of the cephmonitor role can be configured and managed from within the node or category that runs Ceph monitoring.

Example

```
[bright92]% device use bright92
[bright92-->device[bright92]]% roles ; use cephmonitor
[ceph-->device[bright92]-->roles[cephmonitor]]% show
Parameter                        Value
-------------------------------- ----------------------------------
... Extra config parameters...    ...
Monitor data                      /var/lib/ceph/mon/$cluster-$hostname
Name                              cephmonitor
Provisioning associations         <0 internally used>
Revision                          
Type                              CephMonitorRole
```

Ceph monitoring extraconfigparameters setting: Ceph monitoring can also have extra configurations set via the extraconfigparameters option, in a similar way to how it is done for Ceph general configuration (page 508).

The Bright View equivalent to access the preceding cmsh Monitor configuration setting is via the role for a particular node or category. The clickpath that brings up these configuration options for node node004 is, for example:

Devices→Physical Nodes→node004→Edit→Settings→Roles→cephmonitor→Edit

Ceph bootstrap

For completeness, the bootstrap command within ceph mode can be used by the administrator to initialize Ceph Monitors on specified nodes if they are not already initialized. Administrators are however not expected to use it, because they are expected to use the cm-ceph-setup installer utility when installing Ceph in the first place. The installer utility carries out the bootstrap initialization as part of its tasks. The bootstrap command is therefore only intended for use in the unusual case where the administrator would like to set up Ceph storage without using the cm-ceph-setup utility.
10.5 RADOS GW Installation, Initialization, And Properties

10.5.1 RADOS GW Installation and Initialization
If Ceph has been installed using `cm-ceph-setup`, then RADOS is installed and initialized so that it provides a REST API, called the RADOS GW service.

10.5.2 Setting RADOS GW Properties

RADOS GW Properties In cmsh
RADOS GW properties can be managed in cmsh by selecting the device, then assigning the `radosgateway` role to the device. The properties of the role can then be seen and altered:

```
[bright92] device use node004
[bright92->device[node004]]% roles
[bright92->device[node004]->roles] assign radosgateway; commit
[bright92->device[node004]->roles[radosgateway]]% show
```

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>Name</td>
<td>radosgateway</td>
</tr>
<tr>
<td>Provisioning associations</td>
<td>&lt;0 internally used</td>
</tr>
<tr>
<td>Revision</td>
<td></td>
</tr>
<tr>
<td>Type</td>
<td>RadosGatewayRole</td>
</tr>
<tr>
<td>Server Port</td>
<td>7480</td>
</tr>
<tr>
<td>Enable Keystone Authentication</td>
<td>yes</td>
</tr>
<tr>
<td>Keystone Accepted Roles</td>
<td></td>
</tr>
<tr>
<td>Keystone Revocation Interval</td>
<td>600</td>
</tr>
<tr>
<td>Keystone Tokens Cache Size</td>
<td>500</td>
</tr>
<tr>
<td>NSS DB Path</td>
<td>/var/lib/ceph/nss</td>
</tr>
</tbody>
</table>

The role properties can also be accessed via category roles and configuration overlay roles.

- Keystone authentication can be disabled or enabled using `cmsh` to set the `enablekeystoneauthentication` property.

  For example, setting `enablekeystoneauthentication` to `no` on a RADOS GW node, and committing it makes RADOS GW services unavailable to that node.

Example

```
[bright92->device[node004]->roles[radosgateway]]% set enablekeystoneauthentication no
[bright92->device*[@node004*]->roles*[radosgateway*]]% commit
```

RADOS GW Properties In Bright View
RADOS GW properties can be accessed in Bright View via:

- via the node clickpath:
  Devices → Nodes → Settings → Roles → Rados Gateway Role

- via the node category clickpath:
  Grouping → Node Categories → Settings → Roles → Add → Rados Gateway Role

- or via the configuration overlay clickpath:
  Configuration Overlays → Edit → Roles → radosgateway
11 BeeGFS

11.1 BeeGFS Introduction

11.1.1 BeeGFS Concepts

BeeGFS is a high-performance parallel file system, developed by the Fraunhofer Competence Center for High Performance Computing, and optimized for intensive I/O. It uses a distributed metadata architecture designed for scalability and flexibility. More information about BeeGFS can be found at the official online documentation at https://www.beegfs.io/content/documentation/.

NVIDIA Bright Cluster Manager provides packages to allow BeeGFS to be deployed, managed, and monitored on a cluster running RHEL and derivatives. The deployment tool provided by the cluster manager is cm-beegfs-setup (section 11.2).

A BeeGFS cluster consists of a management server, one or more metadata servers and one or more storage servers. Clients should have client servers running on them.

If high-availability is required, then the administrator needs to arrange for it separately, for the BeeGFS management servers.

Since BeeGFS is a parallel filesystem, it means that adding more nodes increases not only the storage capacity, but also system performance. BeeGFS can work with any local file system for data storage as long as it is POSIX compliant.

Versions of BeeGFS before 7.2 had an optional graphical user interface called Admon which was integrated with the cluster manager. This has been dropped since BeeGFS version 7.2, which means it has been dropped since NVIDIA Bright Cluster Manager 9.1. The BeeGFS Mon monitoring service is on the roadmap for future cluster manager integration.

11.1.2 BeeGFS Installation Notes And Options

By default all logging is done to the systemd journal. The installation logs can be viewed using:

```
journalctl -u 'beegfs-*'.
```

Log locations can be changed using cmsh or Bright View at any moment after installation, by editing BeeGFS roles for the corresponding configuration overlays (section 11.2.1).

Authentication is possible for the BeeGFS cluster via a shared secret file. The shared secret file can be installed during cm-beegfs-setup installation (section 11.2).

11.2 Deployment And Uninstallation Of BeeGFS With cm-beegfs-setup

Deployment and uninstallation for the cluster manager can be carried out with the Ncurses-based cm-beegfs-setup utility. The utility is a part of a cluster-tools package that comes with the cluster manager. If the BeeGFS packages are not installed, then the utility installs them.
The `cm-beegfs-setup` utility can be run as root from the head node. The first Ncurses screen that shows up is the main menu for operations (figure 11.1)

![Figure 11.1: BeeGFS Script Main Menu](image)

In the main menu, the administrator can choose to:

- Deploy BeeGFS (section 11.2.1)
- Uninstall BeeGFS (section 11.2.2)
- Simply exit the Ncurses session

### 11.2.1 Deployment Of BeeGFS

If the `Deploy` option is chosen, then the deployment wizard starts.

#### Configuration Overlays

The wizard first asks for names for the configuration overlays (figure 11.2):

![Figure 11.2: Configuration Overlays Creation](image)

These overlay names are for:

- the BeeGFS metadata configuration overlay
- the BeeGFS storage configuration overlay
- the BeeGFS client configuration overlay

#### Select Management Node

The next screen (figure 11.3) then asks for a node to be selected to be the BeeGFS management server:
11.2 Deployment And Uninstallation Of BeeGFS With `cm-beegfs-setup`

Select Metadata Nodes
The next screen (figure 11.4) then asks for nodes to be selected to be the BeeGFS metadata servers:

Select Storage Nodes
The next screen (figure 11.5) asks for nodes to be selected to be the BeeGFS storage servers:

Select Client Nodes
The next screen (figure 11.6) asks for nodes to be selected to be the client nodes:
Management Server Configuration
The next screen (figure 11.7) asks for some configuration parameters for the BeeGFS management server to be set:

Figure 11.7: Management Server Configuration

The values required are for the:
- Path to a data directory
- Management server TCP port
- Management server UDP port

Metadata Server Configuration
The next screen (figure 11.8) asks for input on configuring the BeeGFS metadata servers:

Figure 11.8: Metadata Server Configuration

The values required are for the:
- Path to a data directory
- Metadata server TCP port
- Metadata server UDP port
Storage Server Configuration
The next screen (figure 11.9) asks for input on setting some configuration parameters for the BeeGFS storage servers:

The items required are for the:
- Path to a data directory
- Storage server TCP port
- Storage server UDP port

Client Configuration
The next screen (figure 11.10) asks for input on configuring the BeeGFS clients:

The inputs required are for the:
- Client UDP port
- Helper TCP port

Authentification Configuration
In the next screen, (figure 11.11) a path must be specified for the shared secret file:

This is a file with arbitrary content. If it is set, then it must exist on all BeeGFS nodes.

Deployment Summary
Finally, the wizard displays a deployment summary (figure 11.12):
From this screen the deployment process can be started, the deployment configuration can be viewed and saved, or the wizard can simply be exited.

**Command Line Installation**
If a configuration file is saved from a `cm-beegfs-setup` run, then it can be used to install a BeeGFS cluster automatically:

Example

```
cm-beegfs-setup -c cm-beegfs-setup.conf
```

### 11.2.2 Uninstalling BeeGFS

If the **Uninstall** option is chosen from the main menu (figure 11.1), then the uninstallation process starts.

An extra **Are you sure?** prompt is thrown in the way first, to help avoid a mistaken removal. After confirmation, a BeeGFS instance must then be selected for the removal, by selecting its management node (figure 11.13):

**Uninstall Management Node Selection**

The uninstallation process then starts. It can take several minutes in order to ensure that services are stopped properly, so that errors and junk files are avoided.

### 11.3 Managing The Deployed BeeGFS Instance

A BeeGFS cluster consists of:

- One Management node
- Some Metadata nodes
- Some Storage nodes

Information about the right number of nodes to use can be found at [https://www.beegfs.io/docs/whitepapers/Picking_the_right_Number_of_Targets_per_Server_for_BeeGFS_by_ThinkParQ.pdf](https://www.beegfs.io/docs/whitepapers/Picking_the_right_Number_of_Targets_per_Server_for_BeeGFS_by_ThinkParQ.pdf)
BeeGFS clients should have client servers running on them.

11.3.1 Setup
Associated with BeeGFS are the following BeeGFS-related packages, most of which provide services:

1. beegfs-mgmtd
2. beegfs-meta
3. beegfs-storage
4. beegfs-client
5. beegfs-helperd
6. beegfs-utils

The repository from which these packages are picked up is added to the repositories list when BeeGFS is installed by the cluster manager (section 11.2), and the packages are then installed on all of the cluster nodes. The services are assignable by the cluster administrator to cluster devices as roles. In cmsh these BeeGFS roles are treated as objects, and the configuration of these roles is determined by role parameters (object attributes). The roles are described in detail section 11.3.2.

11.3.2 BeeGFS Objects
BeeGFSManagement Role
The BeeGFSManagement role is used to configure BeeGFS management services. The role parameters are described in table 11.1:

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
<th>Option to beegfs-mgmtd</th>
</tr>
</thead>
<tbody>
<tr>
<td>Data directory</td>
<td>Path to data directory</td>
<td>storeMgmtdDirectory</td>
</tr>
<tr>
<td>Allow new servers</td>
<td>Allow new servers registration</td>
<td>sysAllowNewServers</td>
</tr>
<tr>
<td>Allow new storage targets</td>
<td>Allow new storage targets registration</td>
<td>sysAllowNewTargets</td>
</tr>
<tr>
<td>Authentication file</td>
<td>Path to shared secret authentication file</td>
<td>connAuthFile</td>
</tr>
<tr>
<td>Backlog TCP</td>
<td>TCP listen backlog size</td>
<td>connBacklogTCP</td>
</tr>
<tr>
<td>Interfaces file</td>
<td>Path to file with interfaces list</td>
<td>connInterfacesFile</td>
</tr>
<tr>
<td>Management port TCP</td>
<td>TCP port for management service</td>
<td>connMgmtdPortTCP</td>
</tr>
</tbody>
</table>

...continues
<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
<th>Option to beegfs-mgstd</th>
</tr>
</thead>
<tbody>
<tr>
<td>Management port UDP</td>
<td>UDP port for management service</td>
<td>connMgmtdPortUDP</td>
</tr>
<tr>
<td>Net filter file</td>
<td>Path to file with list of allowed IP subnets</td>
<td>connNetFilterFile</td>
</tr>
<tr>
<td>Log level</td>
<td>Log level</td>
<td>logLevel</td>
</tr>
<tr>
<td>No log date</td>
<td>Do not show date along with time in log</td>
<td>logNoDate</td>
</tr>
<tr>
<td>Number of log lines</td>
<td>Number of lines in log file, after which it will be rotated</td>
<td>logNumLines</td>
</tr>
<tr>
<td>Number of log rotated files</td>
<td>Number of old log files to keep</td>
<td>logNumRotatedFiles</td>
</tr>
<tr>
<td>Log file</td>
<td>Path to log file, empty means logs go to the journal</td>
<td>logStdFile</td>
</tr>
<tr>
<td>Enable Quota</td>
<td>Enable quota</td>
<td>quotaEnableEnforcement</td>
</tr>
<tr>
<td>Quota query GID file</td>
<td>Path to file with GIDs to be checked by quota</td>
<td>quotaQueryGIDFile</td>
</tr>
<tr>
<td>Quota query GID range</td>
<td>GID range to be checked by quota</td>
<td>quotaQueryGIDRange</td>
</tr>
<tr>
<td>Quota query UID file</td>
<td>Path to file with UIDs to be checked by quota</td>
<td>quotaQueryUIDFile</td>
</tr>
<tr>
<td>Quota query UID range</td>
<td>UID range to be checked by quota</td>
<td>quotaQueryUIDRange</td>
</tr>
<tr>
<td>Quota query type</td>
<td>Query type for quota</td>
<td>quotaQueryType</td>
</tr>
<tr>
<td>Quota query with system users groups</td>
<td>Allow also system users/groups to be checked by quota</td>
<td>quotaQueryWithSystemUsersGroups</td>
</tr>
<tr>
<td>Quota update interval</td>
<td>Quota update interval</td>
<td>quotaUpdateIntervalMin</td>
</tr>
</tbody>
</table>

...continues
### Table 11.1: BeeGFSManagement role parameters and `beegfs-mgstd` options

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
<th>Option to <code>beegfs-mgstd</code></th>
</tr>
</thead>
<tbody>
<tr>
<td>Target offline timeout</td>
<td>Timeout for targets on storage server to be considered offline when no target status received</td>
<td><code>sysTargetOfflineTimeoutSecs</code></td>
</tr>
<tr>
<td>Client auto remove time</td>
<td>Time after which an unreachable node is considered dead</td>
<td><code>tuneClientAutoRemoveMins</code></td>
</tr>
<tr>
<td>Number of workers</td>
<td>Number of worker threads</td>
<td><code>tuneNumWorkers</code></td>
</tr>
<tr>
<td>Meta dynamic pools</td>
<td>Raise lower limits if difference in capacity becomes too large between targets</td>
<td><code>tuneMetaDynamicPools</code></td>
</tr>
<tr>
<td>Meta inodes low limit</td>
<td>Metadata inode free space pool threshold</td>
<td><code>tuneMetaInodesLowLimit</code></td>
</tr>
<tr>
<td>Meta inodes emergency limit</td>
<td>Metadata inode free space pool threshold</td>
<td><code>tuneMetaInodesEmergencyLimit</code></td>
</tr>
<tr>
<td>Meta space low limit</td>
<td>Metadata space free space pool threshold</td>
<td><code>tuneMetaSpaceLowLimit</code></td>
</tr>
<tr>
<td>Meta space emergency limit</td>
<td>Metadata space free space pool threshold</td>
<td><code>tuneMetaSpaceEmergencyLimit</code></td>
</tr>
<tr>
<td>Storage dynamic pools</td>
<td>Raise lower limits if difference in capacity becomes too large between targets</td>
<td><code>tuneStorageDynamicPools</code></td>
</tr>
<tr>
<td>Storage inodes low limit</td>
<td>Storage inode free space pool threshold</td>
<td><code>tuneStorageInodesLowLimit</code></td>
</tr>
<tr>
<td>Storage inodes emergency limit</td>
<td>Storage inode free space pool threshold</td>
<td><code>tuneStorageInodesEmergencyLimit</code></td>
</tr>
<tr>
<td>Storage space low limit</td>
<td>Storage space free space pool threshold</td>
<td><code>tuneStorageSpaceLowLimit</code></td>
</tr>
<tr>
<td>Storage space emergency limit</td>
<td>Storage space free space pool threshold</td>
<td><code>tuneStorageSpaceEmergencyLimit</code></td>
</tr>
</tbody>
</table>
The beegfs-mgmtd settings are updated by the cluster manager in `/etc/beegfs/beegfs-mgmtd.conf`.

The settings can be managed in `cmsh` via the `beegfs::management` role. Roles are, as usual, assigned via `device`, `category`, or `configurationoverlay` mode. The parameters of the role that correspond to the parameters in the preceding table can then be set.

Example

```
[bright92 ~]# cmsh
[bright92] % device roles node001
[bright92->device[node001]->roles] % assign beegfs::client beegfs::management beegfs::metadata beegfs::storage
[bright92->device[node001]->roles] % assign beegfs::management
[bright92->device[node001]->roles[BeeGFS::Management]] % show
```

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>Add services</td>
<td>yes</td>
</tr>
<tr>
<td>Name</td>
<td>BeeGFS::Management</td>
</tr>
<tr>
<td>Provisioning associations</td>
<td>&lt;0 internally used&gt;</td>
</tr>
<tr>
<td>Revision</td>
<td>BeeGFSManagementRole</td>
</tr>
<tr>
<td>Allow new servers</td>
<td>yes</td>
</tr>
<tr>
<td>Data directory</td>
<td><code>/var/lib/beegfs/management</code></td>
</tr>
<tr>
<td>Allow new storage targets</td>
<td>yes</td>
</tr>
<tr>
<td>Authentication file</td>
<td></td>
</tr>
<tr>
<td>Backlog TCP</td>
<td>128</td>
</tr>
<tr>
<td>Interfaces file</td>
<td></td>
</tr>
<tr>
<td>Management port TCP</td>
<td>8008</td>
</tr>
<tr>
<td>Management port UDP</td>
<td>8008</td>
</tr>
<tr>
<td>Net filter file</td>
<td></td>
</tr>
<tr>
<td>Log file</td>
<td></td>
</tr>
<tr>
<td>Log level</td>
<td>2</td>
</tr>
<tr>
<td>No log date</td>
<td>no</td>
</tr>
<tr>
<td>Number of log lines</td>
<td>50000</td>
</tr>
<tr>
<td>Number of log rotated files</td>
<td>5</td>
</tr>
<tr>
<td>Enable Quota</td>
<td>no</td>
</tr>
<tr>
<td>Quota query GID file</td>
<td></td>
</tr>
<tr>
<td>Quota query GID range</td>
<td></td>
</tr>
<tr>
<td>Quota query UID file</td>
<td></td>
</tr>
<tr>
<td>Quota query UID range</td>
<td></td>
</tr>
<tr>
<td>Quota query type</td>
<td>system</td>
</tr>
<tr>
<td>Quota query with system users groups</td>
<td>no</td>
</tr>
<tr>
<td>Quota update interval</td>
<td>10m</td>
</tr>
<tr>
<td>Target offline timeout</td>
<td>3m</td>
</tr>
<tr>
<td>Client auto remove time</td>
<td>30m</td>
</tr>
<tr>
<td>Number of workers</td>
<td>4</td>
</tr>
<tr>
<td>Meta dynamic pools</td>
<td>yes</td>
</tr>
<tr>
<td>Meta inodes emergency limit</td>
<td>1M</td>
</tr>
<tr>
<td>Meta inodes low limit</td>
<td>10M</td>
</tr>
<tr>
<td>Meta space emergency limit</td>
<td>3G</td>
</tr>
<tr>
<td>Meta space low limit</td>
<td>10G</td>
</tr>
<tr>
<td>Storage dynamic pools</td>
<td>yes</td>
</tr>
<tr>
<td>Storage inodes emergency limit</td>
<td>1M</td>
</tr>
<tr>
<td>Storage inodes low limit</td>
<td>10M</td>
</tr>
<tr>
<td>Storage space emergency limit</td>
<td>20G</td>
</tr>
<tr>
<td>Storage space low limit</td>
<td>1T</td>
</tr>
</tbody>
</table>
### BeeGFSMetadata Role

The BeeGFSMetadata role is used for configuring BeeGFS metadata services. The role parameters are described in Table 11.2:

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
<th>Option to beegfs-meta</th>
</tr>
</thead>
<tbody>
<tr>
<td>Data directory</td>
<td>Path to the data directory</td>
<td>storeMetaDirectory</td>
</tr>
<tr>
<td>Management node</td>
<td>Node with a management role</td>
<td>sysMgmtdHost</td>
</tr>
<tr>
<td>Authentication file</td>
<td>Path to the shared secret authentication file</td>
<td>connAuthFile</td>
</tr>
<tr>
<td>Backlog TCP</td>
<td>TCP listen backlog</td>
<td>connBacklogTCP</td>
</tr>
<tr>
<td>Fallback expiration</td>
<td>Time after which a connection to a fallback interface expires</td>
<td>connFallbackExpirationSecs</td>
</tr>
<tr>
<td>Interfaces file</td>
<td>Path to the file with a list of interfaces for communication</td>
<td>connInterfacesFile</td>
</tr>
<tr>
<td>Max internode number</td>
<td>Max number of simultaneous connections to the same node</td>
<td>connMaxInternodeNum</td>
</tr>
<tr>
<td>Meta port TCP</td>
<td>TCP port for metadata service</td>
<td>connMetaPortTCP</td>
</tr>
<tr>
<td>Meta port UDP</td>
<td>UDP port for metadata service</td>
<td>connMetaPortUDP</td>
</tr>
<tr>
<td>Net filter file</td>
<td>Path to a file with a list of allowed IP subnets</td>
<td>connNetFilterFile</td>
</tr>
<tr>
<td>Use RDMA</td>
<td>Use RDMA</td>
<td>connUseRDMA</td>
</tr>
<tr>
<td>RDMA type of service</td>
<td>RDMA type of service</td>
<td>connRDMATypeOfService</td>
</tr>
<tr>
<td>Tcp only filter file</td>
<td>Path to a file with a list of no DRMA IP ranges</td>
<td>connTcpOnlyFilterFile</td>
</tr>
<tr>
<td>Log level</td>
<td>Log level</td>
<td>logLevel</td>
</tr>
</tbody>
</table>

…continues
<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
<th>Option to beegfs-meta</th>
</tr>
</thead>
<tbody>
<tr>
<td>No log date</td>
<td>Do not show date along with time in log</td>
<td>logNoDate</td>
</tr>
<tr>
<td>Number of log lines</td>
<td>Number of lines in log file, after which it will be rotated</td>
<td>logNumLines</td>
</tr>
<tr>
<td>Number of log rotated files</td>
<td>Number of old log files to keep</td>
<td>logNumRotatedFiles</td>
</tr>
<tr>
<td>Log file</td>
<td>Path to the log file, empty means logs go to the journal</td>
<td>logStdFile</td>
</tr>
<tr>
<td>Client xattrs</td>
<td>Enable client-side extended attributes</td>
<td>storeClientXAttrs</td>
</tr>
<tr>
<td>Client ACLs</td>
<td>Enable handling and storage of client-side ACLs</td>
<td>storeClientACLs</td>
</tr>
<tr>
<td>Use extended attributes</td>
<td>Store metadata as extended attributes or not</td>
<td>storeUseExtendedAttrs</td>
</tr>
<tr>
<td>Target attachment file</td>
<td>File with a list of targets to be grouped within the same domain for randominternode</td>
<td>sysTargetAttachmentFile</td>
</tr>
<tr>
<td>Target offline timeout</td>
<td>Timeout for targets on a storage server to be considered offline when no target status is received</td>
<td>sysTargetOfflineTimeoutSecs</td>
</tr>
<tr>
<td>Allow user set pattern</td>
<td>Allow non-privileged users to modify stripe pattern settings for directories they own</td>
<td>sysAllowUserSetPattern</td>
</tr>
<tr>
<td>Bind to numa zone</td>
<td>Zero-based NUMA zone number to which all threads of metadata process should be bound</td>
<td>tuneBindToNumaZone</td>
</tr>
<tr>
<td>Number of stream listeners</td>
<td>The number of threads waiting for incoming data events</td>
<td>tuneNumStreamListeners</td>
</tr>
<tr>
<td>Number of workers</td>
<td>Number of worker threads</td>
<td>tuneNumWorkers</td>
</tr>
</tbody>
</table>
11.3 Managing The Deployed BeeGFS Instance

...continued

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
<th>Option to beegfs-meta</th>
</tr>
</thead>
<tbody>
<tr>
<td>Target chooser</td>
<td>The algorithm to choose storage targets for file creation</td>
<td>tuneTargetChooser</td>
</tr>
<tr>
<td>Use aggressive</td>
<td>Actively poll for events instead of sleeping until an event occurs</td>
<td>tuneUseAggressiveStreamPoll</td>
</tr>
<tr>
<td>stream poll</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Use per user msg</td>
<td>Use per-user queues for pending requests</td>
<td>tuneUsePerUserMsgQueues</td>
</tr>
<tr>
<td>queues</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

Table 11.2: BeeGFSMetadata role parameters and beegfs-meta options

The beegfs-meta settings are updated by the cluster manager in /etc/beegfs/beegfs-meta.conf. The settings can be managed in cmsh via the beegfs::metadata role. Roles are, as usual, assigned via device, category, or configurationoverlay mode. The parameters of the role that correspond to the parameters in the preceding table can then be set.

Example

```
[bright92 ~]$ cmsh
[bright92]$ category roles default
[bright92->category[default]->roles]$ assign bee<TAB><TAB><TAB>
beegfs::client beegfs::management beegfs::metadata beegfs::storage
[bright92->category[default]->roles]$ assign beegfs::metadata
[bright92->category[default]->roles[BeeGFS::Metadata]]$ show

Parameter                     Value
----------------------------------------------------------------------------------
Add services                  yes
Name                           BeeGFS::Metadata
Provisioning associations     <0 internally used>
Revision                      
Type                           BeeGFSMetadataRole
Data directory                /var/lib/beegfs/metadata
Management node               
Authentication file            
Backlog TCP                   128
Fallback expiration           15m
Interfaces file               
Max internode number          32
Meta port TCP                 8005
Meta port UDP                 8005
Net filter file               
RDMA type of service          0
Tcp only filter file          
Use RDMA                      yes
Log file                      
Log level                     3
No log date                   no
Number of log lines           50000
Number of log rotated files   5
Client ACLs                   no
```
Client xattrs: no
Use extended attributes: yes
Allow user set pattern: no
Target attachment file: 
Target offline timeout: 3m
Bind to numa zone: 
Number of stream listeners: 1
Number of workers: 0
Target chooser: randomized
Use aggressive stream poll: no
Use per user msg queues: no

**BeeGFSStorage Role**

The BeeGFSStorage role is used for configuring BeeGFS storage services. The role parameters are described in Table 11.3:

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
<th>Option to beegfs-storage</th>
</tr>
</thead>
<tbody>
<tr>
<td>Data directories</td>
<td>Path to the data directories</td>
<td>storeStorageDirectory</td>
</tr>
<tr>
<td>Management node</td>
<td>Node with a management role</td>
<td>sysMgmtdHost</td>
</tr>
<tr>
<td>Authentication file</td>
<td>Path to the shared secret authentification file</td>
<td>connAuthFile</td>
</tr>
<tr>
<td>Backlog TCP</td>
<td>TCP listen backlog</td>
<td>connBacklogTCP</td>
</tr>
<tr>
<td>Interfaces file</td>
<td>Path to the file with a list of interfaces for communication</td>
<td>connInterfacesFile</td>
</tr>
<tr>
<td>Max internode number</td>
<td>Max number of simultaneous connections to the same node</td>
<td>connMaxInternodeNum</td>
</tr>
<tr>
<td>Storage port TCP</td>
<td>TCP port for storage service</td>
<td>connStoragePortTCP</td>
</tr>
<tr>
<td>Storage port UDP</td>
<td>UDP port for storage service</td>
<td>connStoragePortUDP</td>
</tr>
<tr>
<td>Net filter file</td>
<td>Path to a file with a list of allowed IP subnets</td>
<td>connNetFilterFile</td>
</tr>
<tr>
<td>Use RDMA</td>
<td>Use RDMA</td>
<td>connUseRDMA</td>
</tr>
</tbody>
</table>

...continues
### 11.3 Managing The Deployed BeeGFS Instance

#### Parameters and Options

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
<th>Option to beegfs-storage</th>
</tr>
</thead>
<tbody>
<tr>
<td>RDMA type of service</td>
<td>RDMA type of service</td>
<td>connRDMATypeOfService</td>
</tr>
<tr>
<td>Tcp only filter file</td>
<td>Path to a file with a list of subnets with no RDMA</td>
<td>connTcpOnlyFilterFile</td>
</tr>
<tr>
<td>Log level</td>
<td>Log level</td>
<td>logLevel</td>
</tr>
<tr>
<td>No log date</td>
<td>Do not show date along with time in log</td>
<td>logNoDate</td>
</tr>
<tr>
<td>Number of log lines</td>
<td>Number of lines in log file, after which it will be rotated</td>
<td>logNumLines</td>
</tr>
<tr>
<td>Number of log rotated files</td>
<td>Number of old log files to keep</td>
<td>logNumRotatedFiles</td>
</tr>
<tr>
<td>Log file</td>
<td>Path to the log file, empty means logs go to the journal</td>
<td>logStdFile</td>
</tr>
<tr>
<td>Resync safety threshold</td>
<td>Add an extra amount of time to the last successful communication timestamp in case of a potential cache loss</td>
<td>sysResyncSafetyThresholdMins</td>
</tr>
<tr>
<td>Target offline timeout</td>
<td>Timeout until targets on a storage server are considered offline when no target status is received</td>
<td>sysTargetOfflineTimeoutSecs</td>
</tr>
<tr>
<td>Bind to numa zone</td>
<td>Zero-based NUMA zone number to which all threads of metadata process should be bound</td>
<td>tuneBindToNumaZone</td>
</tr>
<tr>
<td>File read ahead size</td>
<td>Byte range submitted to the kernel for read-ahead after number of bytes was already read from target</td>
<td>tuneFileReadAheadSize</td>
</tr>
<tr>
<td>File read ahead trigger size</td>
<td>Number of bytes read after which the read-ahead is triggered</td>
<td>tuneFileReadAheadTriggerSize</td>
</tr>
<tr>
<td>File read size</td>
<td>Maximum amount of data server should read in a single operation</td>
<td>tuneFileReadSize</td>
</tr>
</tbody>
</table>

...continues
<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
<th>Option to beegfs-storage</th>
</tr>
</thead>
<tbody>
<tr>
<td>File write size</td>
<td>Maximum amount of data server should write in a single operation</td>
<td>tuneFileWriteSize</td>
</tr>
<tr>
<td>File write sync size</td>
<td>Number of bytes after which kernel advised to commit data</td>
<td>tuneFileWriteSyncSize</td>
</tr>
<tr>
<td>Number of resync gather slaves</td>
<td>Number of threads to gather filesystem information for a buddy mirror resync</td>
<td>tuneNumResyncGatherSlaves</td>
</tr>
<tr>
<td>Number of resync slaves</td>
<td>Number of threads to sync filesystem information for a buddy mirror resync</td>
<td>tuneNumResyncSlaves</td>
</tr>
<tr>
<td>Number of stream listeners</td>
<td>Number of threads waiting for incoming data events</td>
<td>tuneNumStreamListeners</td>
</tr>
<tr>
<td>Number of workers</td>
<td>Number of worker threads</td>
<td>tuneNumWorkers</td>
</tr>
<tr>
<td>Use aggressive stream poll</td>
<td>Actively poll for events instead of sleeping until an event occur</td>
<td>tuneUseAggressiveStreamPoll</td>
</tr>
<tr>
<td>Use per target workers</td>
<td>Create a separate set of workers and attach it for each storage target</td>
<td>tuneUsePerTargetWorkers</td>
</tr>
<tr>
<td>Use per user msg queues</td>
<td>Use per-user queues for pending requests</td>
<td>tuneUsePerUserMsgQueues</td>
</tr>
<tr>
<td>Worker buffer size</td>
<td>Size of network and io buffers, allocated for each worker</td>
<td>tuneWorkerBufSize</td>
</tr>
</tbody>
</table>

Table 11.3: BeeGFSStorage role parameters and beegfs-storage options

The beegfs-storage settings are updated by the cluster manager in `/etc/beegfs/beegfs-storage.conf`. The settings can be managed in cmsh via the beegfs::storage role. Roles are, as usual, assigned via device, category, or configurationoverlay mode. The parameters of the role that correspond to the parameters in the preceding table can then be set.

Example

```
[bright92 ~]# cmsh
[bright92] % category roles default
[bright92->category[default]->roles] % assign beegfs::client beegfs::management beegfs::metadata beegfs::storage
[bright92->category[default]->roles] % assign beegfs::storage
[bright92->category[*][default*]->roles*[BeeGFS::Storage*]] % show
```
### Parameter

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
<th>Option to beegfs-client</th>
</tr>
</thead>
<tbody>
<tr>
<td>Add services</td>
<td>yes</td>
<td></td>
</tr>
<tr>
<td>Name</td>
<td>BeeGFS::Storage</td>
<td></td>
</tr>
<tr>
<td>Provisioning associations</td>
<td>&lt;0 internally used&gt;</td>
<td></td>
</tr>
<tr>
<td>Revision</td>
<td>BeeGFSStorageRole</td>
<td></td>
</tr>
<tr>
<td>Data directories</td>
<td>/var/lib/beegfs/storage</td>
<td></td>
</tr>
<tr>
<td>Management node</td>
<td>Authentication file</td>
<td></td>
</tr>
<tr>
<td>Backlog TCP</td>
<td>128</td>
<td></td>
</tr>
<tr>
<td>Interfaces file</td>
<td>Max internode number</td>
<td>12</td>
</tr>
<tr>
<td>Net filter file</td>
<td>RDMA type of service</td>
<td>0</td>
</tr>
<tr>
<td>Storage port TCP</td>
<td>8003</td>
<td></td>
</tr>
<tr>
<td>Storage port UDP</td>
<td>8003</td>
<td></td>
</tr>
<tr>
<td>Tcp only filter file</td>
<td>Use RDMA</td>
<td>yes</td>
</tr>
<tr>
<td>Log file</td>
<td>Log level</td>
<td>3</td>
</tr>
<tr>
<td>No log date</td>
<td>Number of log lines</td>
<td>50000</td>
</tr>
<tr>
<td>Number of log rotated files</td>
<td>Resync safety threshold</td>
<td>10m</td>
</tr>
<tr>
<td>Target offline timeout</td>
<td>Bind to numa zone</td>
<td></td>
</tr>
<tr>
<td>File read ahead size</td>
<td>File read ahead trigger size</td>
<td>4m</td>
</tr>
<tr>
<td>File read size</td>
<td>File write size</td>
<td>128k</td>
</tr>
<tr>
<td>File write sync size</td>
<td>Number of resync gather slaves</td>
<td>6</td>
</tr>
<tr>
<td>Number of resync slaves</td>
<td>Number of stream listeners</td>
<td>1</td>
</tr>
<tr>
<td>Number of workers</td>
<td>Use aggressive stream poll</td>
<td>no</td>
</tr>
<tr>
<td>Use per target workers</td>
<td>Use per user msg queues</td>
<td>no</td>
</tr>
<tr>
<td>Worker buffer size</td>
<td>4m</td>
<td></td>
</tr>
</tbody>
</table>

### BeeGFSClient Role

The BeeGFSClient role is used for configuring BeeGFS client and helperd services. The role parameters are described in table 11.4 and in table 11.5:
<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
<th>Option to beegfs-client</th>
</tr>
</thead>
<tbody>
<tr>
<td>Authentication file</td>
<td>Path to the shared secret authentication file</td>
<td>connAuthFile</td>
</tr>
<tr>
<td>Client port UDP</td>
<td>UDP port for client service</td>
<td>connClientPortUDP</td>
</tr>
<tr>
<td>Helper port TCP</td>
<td>TCP port for helper service</td>
<td>connHelperdPortTCP</td>
</tr>
<tr>
<td>Communication retry time</td>
<td>Time for retries in case of a network failure</td>
<td>connCommRetrySecs</td>
</tr>
<tr>
<td>Fallback expiration time</td>
<td>Time after which a connection to a fallback interface expires</td>
<td>connFallbackExpirationSecs</td>
</tr>
<tr>
<td>Interfaces file</td>
<td>Path to file with list of communication interfaces</td>
<td>connInterfacesFile</td>
</tr>
<tr>
<td>Max internode number</td>
<td>Maximum simultaneous connections to the same node</td>
<td>connMaxInternodeNum</td>
</tr>
<tr>
<td>Net filter file</td>
<td>Path to a file with a list of allowed IP subnets</td>
<td>connNetFilterFile</td>
</tr>
<tr>
<td>Use RDMA</td>
<td>Use RDMA</td>
<td>connUseRDMA</td>
</tr>
<tr>
<td>RDMA buffers number</td>
<td>Number of RDMA buffers</td>
<td>connRDMABufNum</td>
</tr>
<tr>
<td>RDMA buffer size</td>
<td>Maximum size of a buffer that will be sent over the network</td>
<td>connRDMABufSize</td>
</tr>
<tr>
<td>RDMA type of service</td>
<td>RDMA type of service</td>
<td>connRDMATypeOfService</td>
</tr>
<tr>
<td>Tcp only filter file</td>
<td>Path to a file with a list of no RDMA IP ranges</td>
<td>connTcpOnlyFilterFile</td>
</tr>
<tr>
<td>Log level</td>
<td>Log level</td>
<td>logLevel</td>
</tr>
<tr>
<td>Enable Quota</td>
<td>Enable quota</td>
<td>quotaEnabled</td>
</tr>
</tbody>
</table>
### 11.3 Managing The Deployed BeeGFS Instance

...continued

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
<th>Option to beegfs-client</th>
</tr>
</thead>
<tbody>
<tr>
<td>Create hardlinks as symlinks</td>
<td>Create a symlink when an application tries to create a hardlink</td>
<td>sysCreateHardlinksAsSymlinks</td>
</tr>
<tr>
<td>Mount sanity check ms</td>
<td>Time in ms that server has to respond after mount sanity check</td>
<td>sysMountSanityCheckMS</td>
</tr>
<tr>
<td>Session check on close</td>
<td>Check for valid sessions on storage server when a file is closed</td>
<td>sysSessionCheckOnClose</td>
</tr>
<tr>
<td>Sync on close</td>
<td>Sync file content on close</td>
<td>sysSyncOnClose</td>
</tr>
<tr>
<td>Target offline timeout</td>
<td>Timeout until all storage targets are considered offline when no target state updates can be fetched from management server</td>
<td>sysTargetOfflineTimeoutSecs</td>
</tr>
<tr>
<td>Update target states time</td>
<td>Interval for storage targets states check</td>
<td>sysUpdateTargetStatesSecs</td>
</tr>
<tr>
<td>Enable xattrs</td>
<td>Enable xattrs</td>
<td>sysXAttrsEnabled</td>
</tr>
<tr>
<td>Enable ACLs</td>
<td>Enable ACLs</td>
<td>sysACLsEnabled</td>
</tr>
<tr>
<td>File cache type</td>
<td>File read/write cache type</td>
<td>tuneFileCacheType</td>
</tr>
<tr>
<td>Preferred meta file</td>
<td>Path to a file with preferred metadata servers</td>
<td>tunePreferredMetaFile</td>
</tr>
<tr>
<td>Preferred storage file</td>
<td>Path to a file with preferred storage targets</td>
<td>tunePreferredStorageFile</td>
</tr>
<tr>
<td>Remote fsync</td>
<td>Should fsync be executed on server to flush cached file</td>
<td>tuneRemoteFSync</td>
</tr>
<tr>
<td>Use global append locks</td>
<td>Should files, opened in append mode, be protected by locks on local machine (false) or on servers (true)</td>
<td>tuneUseGlobalAppendLocks</td>
</tr>
<tr>
<td>Use global file locks</td>
<td>Should advisory locks be checked on local machine (false) or on servers</td>
<td>tuneUseGlobalFileLocks</td>
</tr>
</tbody>
</table>

...continues
The `beegfs-client` settings are updated by the cluster manager in `/etc/beegfs/beegfs-client.conf`.

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
<th>Option to <code>beegfs-client</code></th>
</tr>
</thead>
<tbody>
<tr>
<td>Authentication file</td>
<td>Path to the shared secret</td>
<td><code>connAuthFile</code></td>
</tr>
<tr>
<td>Helper port TCP</td>
<td>TCP port for helper service</td>
<td><code>connHelperdPortTCP</code></td>
</tr>
<tr>
<td>No log date</td>
<td>Do not show date along with time in log</td>
<td><code>logNoDate</code></td>
</tr>
<tr>
<td>Number of log lines</td>
<td>Number of lines in log file, after which it is rotated</td>
<td><code>logNumLines</code></td>
</tr>
<tr>
<td>Number of log rotated files</td>
<td>Number of old log files to keep</td>
<td><code>logNumRotatedFiles</code></td>
</tr>
<tr>
<td>Log file</td>
<td>Path to the log file, empty means logs go to the journal</td>
<td><code>logStdFile</code></td>
</tr>
<tr>
<td>Helper workers number</td>
<td>Number of worker threads for helper service</td>
<td><code>tuneNumWorkers</code></td>
</tr>
</tbody>
</table>

Table 11.5: `BeeGFSClient role parameters and beegfs-helperd options`

The `beegfs-helperd` settings are updated by the cluster manager within `/etc/beegfs/beegfs-helperd.conf`. The settings can be managed in `cmsh` via the `beegfs::client` role:

```
[bright92 ~]# cmsh
[bright92]% category roles default
[bright92->category[default]->roles]% assign beegfs::management beegfs::metadata beegfs::storage beegfs::client
[bright92->category[default]->roles]% assign beegfs::client beegfs::storage
[bright92->category[default]->roles]% assign beegfs::client [bright92->category[default]->roles][BeeGFS::Client%]
```

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>Add services</td>
<td>yes</td>
</tr>
<tr>
<td>Name</td>
<td>BeeGFS::Client</td>
</tr>
<tr>
<td>Provisioning associations</td>
<td>&lt;0 internally used&gt;</td>
</tr>
<tr>
<td>Revision</td>
<td></td>
</tr>
</tbody>
</table>
### 11.3 Managing The Deployed BeeGFS Instance

<table>
<thead>
<tr>
<th>Type</th>
<th>BeeGFSClientRole</th>
</tr>
</thead>
<tbody>
<tr>
<td>Management node</td>
<td></td>
</tr>
<tr>
<td>Authentication file</td>
<td></td>
</tr>
<tr>
<td>Client port UDP</td>
<td>8004</td>
</tr>
<tr>
<td>Communication retry time</td>
<td>10m</td>
</tr>
<tr>
<td>Fallback expiration time</td>
<td>15m</td>
</tr>
<tr>
<td>Helper port TCP</td>
<td>8006</td>
</tr>
<tr>
<td>Interfaces file</td>
<td></td>
</tr>
<tr>
<td>Max internode number</td>
<td>12</td>
</tr>
<tr>
<td>Mountpoint</td>
<td>/mnt/beegfs</td>
</tr>
<tr>
<td>Net filter file</td>
<td></td>
</tr>
<tr>
<td>RDMA buffer size</td>
<td>8192</td>
</tr>
<tr>
<td>RDMA buffers number</td>
<td>70</td>
</tr>
<tr>
<td>RDMA type of service</td>
<td>0</td>
</tr>
<tr>
<td>Tcp only filter file</td>
<td></td>
</tr>
<tr>
<td>Use RDMA</td>
<td>yes</td>
</tr>
<tr>
<td>Log file</td>
<td></td>
</tr>
<tr>
<td>Log level</td>
<td>3</td>
</tr>
<tr>
<td>No log date</td>
<td>no</td>
</tr>
<tr>
<td>Number of log lines</td>
<td>50000</td>
</tr>
<tr>
<td>Number of log rotated files</td>
<td>5</td>
</tr>
<tr>
<td>Enable Quota</td>
<td>no</td>
</tr>
<tr>
<td>Create hardlinks as symlinks</td>
<td>no</td>
</tr>
<tr>
<td>Enable ACLs</td>
<td>no</td>
</tr>
<tr>
<td>Enable xattrs</td>
<td>no</td>
</tr>
<tr>
<td>Mount sanity check ms</td>
<td>11s</td>
</tr>
<tr>
<td>Session check on close</td>
<td>no</td>
</tr>
<tr>
<td>Sync on close</td>
<td>no</td>
</tr>
<tr>
<td>Target offline timeout</td>
<td>15m</td>
</tr>
<tr>
<td>Update target states time</td>
<td>1m</td>
</tr>
<tr>
<td>File cache type</td>
<td>buffered</td>
</tr>
<tr>
<td>Helper workers number</td>
<td>2</td>
</tr>
<tr>
<td>Preferred meta file</td>
<td></td>
</tr>
<tr>
<td>Preferred storage file</td>
<td></td>
</tr>
<tr>
<td>Remote fsync</td>
<td>yes</td>
</tr>
<tr>
<td>Use global append locks</td>
<td>no</td>
</tr>
<tr>
<td>Use global file locks</td>
<td>no</td>
</tr>
</tbody>
</table>

### 11.3.3 Usage

**Accessing The Filesystem**

After the installation process finishes, the BeeGFS filesystem can be accessed on the client nodes or on the head node. The default mount point is `/mnt/beegfs`, which can be changed within `/etc/beegfs/beegfs-mounts.conf`.
12

Post-Installation Software Management

Some time after NVIDIA Bright Cluster Manager has been installed, administrators may wish to manage other software on the cluster. This means carrying out software management actions such as installation, removal, updating, version checking, and so on.

Since the cluster manager is built on top of an existing Linux distribution, it is best that the administrator use distribution-specific package utilities for software 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 Bright Computing repository. Access to the Bright Computing 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/.

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.

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.

This chapter covers the techniques of software management for the cluster.

Section 12.1 describes the naming convention for a cluster manager RPM or .deb package.

Section 12.2 describes how an RPM or .deb package is managed for the head node.

Section 12.3 describes how an RPM or .deb kernel package can be managed on a head node or image.

Section 12.4 describes how an RPM or .deb package can be managed on a software image.

Section 12.5 describes how a software other than an RPM or .deb package can be managed on a software image.

Section 12.6 describes how custom software images are created that are completely independent of the existing software image distribution and version.

Section 12.7 describes how multi-architecture and multi-distribution images can be created.

12.1 NVIDIA Bright 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:
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 Bright Computing for the cluster manager
- `x.y` (9.2) is the version of NVIDIA Bright 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 Bright Computing 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@bright92 ~]# rpm -qf /usr/bin/zless
gzip-1.9-9.el8.x86_64
[root@bright92 ~]# 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**

```
```
As an aside, system administrators should be aware that the Bright Computing 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 Bright Computing version with a distribution-provided version can result in subtle and hard-to-trace problems in the cluster, and Bright Computing 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.rps.org, while APT is documented for Ubuntu at http://manpages.ubuntu.com/manpages/.

12.1.1 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:

```
[bright92] device use node001
[bright92->device[node001]]$ packages -a
```

<table>
<thead>
<tr>
<th>Node</th>
<th>Type</th>
<th>Name</th>
<th>Version</th>
<th>Arch</th>
<th>Size</th>
<th>Install date</th>
</tr>
</thead>
<tbody>
<tr>
<td>node001</td>
<td>deb</td>
<td>accountsservice</td>
<td>0.6.45-1ubuntu1</td>
<td>amd64</td>
<td>440kB</td>
<td>2019/02/14 10:51:06</td>
</tr>
<tr>
<td>node001</td>
<td>deb</td>
<td>acl</td>
<td>2.2.52-3build1</td>
<td>amd64</td>
<td>200kB</td>
<td>2019/02/14 10:51:19</td>
</tr>
<tr>
<td>node001</td>
<td>deb</td>
<td>acpid</td>
<td>1:2.0.28-1ubuntu1</td>
<td>amd64</td>
<td>139kB</td>
<td>2019/02/14 10:51:19</td>
</tr>
<tr>
<td>node001</td>
<td>deb</td>
<td>adduser</td>
<td>3.116ubuntu1</td>
<td>all</td>
<td>624kB</td>
<td>2019/02/14 10:49:53</td>
</tr>
</tbody>
</table>

The -c|--category option can be used to list all the packages installed in a node category:

```
[bright92] device
[bright92->device]$ packages -a -c default
```

<table>
<thead>
<tr>
<th>Node</th>
<th>Type</th>
<th>Name</th>
<th>Version</th>
<th>Arch</th>
<th>Size</th>
<th>Install date</th>
</tr>
</thead>
<tbody>
<tr>
<td>node001</td>
<td>deb</td>
<td>accountsservice</td>
<td>0.6.45-1ubuntu1</td>
<td>amd64</td>
<td>440kB</td>
<td>2019/02/14 10:51:06</td>
</tr>
<tr>
<td>node001</td>
<td>deb</td>
<td>acl</td>
<td>2.2.52-3build1</td>
<td>amd64</td>
<td>200kB</td>
<td>2019/02/14 10:51:19</td>
</tr>
<tr>
<td>node002</td>
<td>deb</td>
<td>accountsservice</td>
<td>0.6.45-1ubuntu1</td>
<td>amd64</td>
<td>440kB</td>
<td>2019/02/14 10:51:06</td>
</tr>
<tr>
<td>node002</td>
<td>deb</td>
<td>acl</td>
<td>2.2.52-3build1</td>
<td>amd64</td>
<td>200kB</td>
<td>2019/02/14 10:51:19</td>
</tr>
</tbody>
</table>
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

```
[d92]\$ device
[d92]->device]$ packages -c default -f cmdaemon
```

<table>
<thead>
<tr>
<th>Node</th>
<th>Type</th>
<th>Name</th>
<th>Version</th>
<th>Arch</th>
<th>Size</th>
<th>Install date</th>
</tr>
</thead>
<tbody>
<tr>
<td>node001</td>
<td>deb</td>
<td>cmdaemon</td>
<td>9.1-146965-cm9.1-e6f593b676 amd64</td>
<td>87MiB</td>
<td>2020/12/23 17:41:24</td>
<td></td>
</tr>
<tr>
<td>node002</td>
<td>deb</td>
<td>cmdaemon</td>
<td>9.1-146965-cm9.1-e6f593b676 amd64</td>
<td>87MiB</td>
<td>2020/12/23 17:41:24</td>
<td></td>
</tr>
<tr>
<td>...</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

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

12.2 Managing Packages On The Head Node

12.2.1 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.

Bright Computing maintains YUM and zypper repositories of its packages at:

- [http://updates.brightcomputing.com/yum](http://updates.brightcomputing.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 Bright Computing .deb package repositories are at:

- [http://updates.brightcomputing.com/deb](http://updates.brightcomputing.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@brightcomputing.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 Bright Computing GPG public key (0x5D849C16), installed by default in `/etc/pki/rpm-gpg/RPM-GPG-KEY-cm` for Red Hat and derivaties. The Bright Computing public key is also listed in Appendix B.

The first time YUM or zypper are used to install updates, the user is asked whether the Bright Computing public key should be imported into the local repository packages database. Before answering
with a “Y”, yum users may choose to compare the contents of `/etc/pki/rpm-gpg/RPM-GPG-KEY-cm` with
the key listed in Appendix B to verify its integrity. Alternatively, the key may be imported into the local
RPM database directly, using the following command:

```
rpm --import /etc/pki/rpm-gpg/RPM-GPG-KEY-cm
```

With APT, the cluster manager keyring is already imported into `/etc/apt/trusted.gpg.d/brightcomputing-archive-gpg` if the `cm-config-apt` package, provided by the Bright Computing repository, has been installed. The
`cm-config-apt` package is installed by default for the Ubuntu edition of the cluster manager.

**Third Party Packages**

Installation of the following third party packages, most of which are repackaged by Bright Computing
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 Bright Computing for instal-
lation 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)
  - FLEXlm License Daemon (section 7.3.4)
- CUDA (section 7.4)

Exclusion of packages on the head node can be carried out as explained in section 12.3.2, where the
kernel package is used as an example for exclusion.

**Gnome And KDM Installation—Disabling network-manager**

`NetworkManager` interferes with the custom network management of the cluster manager and should
thus never be enabled.

When installing the cluster from bare metal using the cluster manager installer, if the X11 graphical
user installation option checkbox is ticked, then `NetworkManager` is disabled by using a finalize script,
so that the cluster manager’s networking is able to work properly.

However, some time after the cluster is configured, the administrator may wish to install the Gnome
or KDE graphical desktop environment. This can be done in YUM or APT. For example in RHEL the
installation can be carried out with:

```
yum groupinstall "GNOME Desktop"
```

The Network Manager package is a dependency of some Gnome and KDE desktop components,
and may therefore automatically be installed by the package manager command. As a result, the
`NetworkManager` service may be enabled, and it may start to run with the distribution defaults. How-
ever, this interferes with the custom network management of the cluster manager, and `NetworkManager`
should therefore be disabled in this case.

Disabling Network Manager can be done as follows:

- On RHEL 7, RHEL 8 and derivatives, and Ubuntu:

  Example
12.3 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.

12.3.1 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 12.2).
• 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 12.4), but with some special aspects that are discussed in this section.

When a kernel is updated or reinstalled (section 12.3.3), 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 12.4, extra considerations for kernel packages are:

• The kernel must also be explicitly set in CMDaemon (section 12.3.3) before it may be used by the regular nodes.

• If using the chroot method to install the kernel rather than the cm-chroot-sw-image 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 12.4.3).

• 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.

12.3.2 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@bright92 ~]# 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@bright92 ~]# 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.

### 12.3.3 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 12.4), 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 Bright View or *cmsh*.

- To enable it using Bright 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 12.1).

![Figure 12.1: Updating A Software Image Kernel With Bright View](image)

- 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@bright92 ~]# cmsh
  [bright92]% softwareimage
  [bright92]->softwareimage% use default-image
  [bright92->softwareimage[default-image]]% set kernelversion 3.10.0-327.3.1.el7.x86_64
  [bright92->softwareimage*[default-image*]]% commit -w
  ```

  Tab-completion suggestions for the *set kernelversion* command will display the available values for the kernel version.

### 12.3.4 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 Bright View, to enable booting with this kernel option setting, the clickpath
Provisioning → Software images → Edit → Settings → Kernel parameters (figure 12.1) 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@bright92 ~]# cmsh
[bright92] % softwareimage
[bright92] -> softwareimage% use default-image
[bright92->softwareimage[default-image]]% append kernelparameters " apm=off"
[bright92->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.

### 12.3.5 Kernel Driver Modules

Bright Computing 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 Bright Computing
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.e17.x86_64 is not installed
Kernel development package kernel-devel-3.10.0-957.1.3.e17.x86_64 not found
If needed, try to install the kernel development package with:
# yum install kernel-devel-3.10.0-957.1.3.e17.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.e17.x86_64 is not installed
Kernel development package kernel-devel-3.10.0-957.1.3.e17.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.e17.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.e17.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. Bright Computing 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 12.5.2), 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 onwards, 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 12.3.5).

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

```bash
[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).
12.4 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.

12.4.1 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-image, 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 chrooting 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@bright92 ~]# chroot /cm/images/default-image
[root@bright92 /]# yum update    # for RHEL variants
...updates happen...
[root@bright92 /]# 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

```
bright92:~# chroot /cm/images/default-image
bright92.cm.cluster:/ # zypper up     # for SLES
...updates happen...
bright92.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-image`, 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-image` command:

Example

```
root@bright92:~# 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@bright92:/# apt update; apt upgrade    # for Ubuntu
...An upgrade session runs in the image root. Some administrator inputs may be needed...
root@bright92:/# exit           # get out of chroot
...messages indicate that the special directories have been unmounted automatically...
```

The `cm-chroot-sw-image` 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:

```bash
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:

```bash
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 12.3.2.

- 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:

```bash
[root@bright92 ]# cd /cm/images/default-image/etc/yum.repos.d/
[root@bright92 yum.repos.d]# yum --installroot=/cm/images/default-image/ 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.

### 12.4.2 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.

### 12.4.3 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**

```bash
[root@bright92 ]# cmsh -c "softwareimage lock default-image"
[root@bright92 ]# yum --installroot /cm/images/default-image update
[root@bright92 ]# 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.

Bright Computing’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 12.3.3) can be inspected using the cluster manager with:

**Example**

```bash
cmsh -c "softwareimage; use default-image; get kernelversion"
```

### 12.4.4 Managing A Package In The Node-Installer Image

A special software image is the node-installer image. The node-installer image was introduced in NVIDIA Bright Cluster Manager version 9.0, to make multiarch (section 12.7) 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:

```bash
chroot /cm/node-installer yum update
```

or
12.5 Managing Non-RPM Software In A Software Image And Running It On Nodes

... (continuation of previous paragraph)

12.5 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 12.5.1)
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 12.5.2)

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 12.6.

However, this current section (12.5) 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 12.2.2, appropriate software components go under:

/cm/shared/
|-- apps
|-- docs
|-- examples
|-- licenses
`-- modulefiles

12.5.1 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 12.4:

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.

### 12.5.2 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 12.5.1, 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" or "Synchronize image" menu option from Bright 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 12.5.1).

- 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:

  ```bash
e commands \n
  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 Bright View

In Bright View, saving the node state from, for example, `node001` to a new image is done by selecting the appropriate menu option from the clickpath `Devices→Nodes[node001]→Edit→Settings→Actions→Software image→option` (figure 12.2).
Managing Non-RPM Software In A Software Image And Running It On Nodes

The possible options for saving a node state to an image are:

1. The **Grab to image** option: This opens up a dialog offering an image to sync to. Selecting the image creates a new image, grabbing 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 grabbing to a new image” list, available under Node Categories via the clickpath **Grouping → Node categories[default-image] → Settings → Exclude list grab new**. The exclude list is known as **excludelistgrabnew** (page 554) in cmsh.

2. The **Synchronize image** option: This does a sync from the node back to the software image that the node is provisioned with, 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” list, available under Node Categories via the clickpath **Grouping → Node categories[default-image] → Settings → Exclude list grab**. This exclude list is known as **excludelistgrab** (page 555) in cmsh.

The synchronize option can be viewed by the administrator as being a more “gentle” way to carry out a node-to-image sync, compared to the grab option—that is, a “gentle sync” that tries to avoid wiping out existing files, versus a “violent grab” that can wipe out existing files.

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.

The images that are available for selection with the **Grab to image** option can be existing images, while the image that the **Synchronize image** option syncs to is the existing image. If such 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 Bright View method can alternatively be carried out using `cmsh` commands. The `cmsh` equivalent to the `Synchronize image` option is the `grabimage` command, available from `device` mode. The `cmsh` equivalent to the `Grab to image` option is the `grabimage -i` command, where the `-i` option specifies the new image it will write to. As before, that image must be created or cloned beforehand.

Cloning an image for use, setting a category of nodes that will use it, and then synchronizing a node that has the new software setup over to the new image on the provisioner might be carried out as follows via `cmsh`:

```
[root@bright92 ~]# cmsh
[bright92]% softwareimage
[bright92->softwareimage]% clone default-image default-image1
[bright92->softwareimage*[default-image1]]% commit
[bright92->softwareimage*[default-image1]]% category
[bright92->category][default-image1]% clone default default1
[bright92->category*[default1]]% commit
[bright92->category*[default1]]% set softwareimage default-image1
[bright92->category*[default1]]% commit
[bright92->category*[default1]]% device
[bright92->device][default-image1% grabimage -w -i default-image1 node001
[bright92->device][default-image1% grabimage -w -i default-image1 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:

- The `excludelistgrabnew` object. This is used with `grabimage` with the `-i` option. The list can be accessed and edited under `cmsh`, in its `category` mode for a node image, as the `excludelistgrabnew` object. It corresponds to the "Exclude list grabbing to a new image" exclusion list associated with the `Grab to image` option (page 553) in Bright View.
12.6 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 12.6.1) is to create a base (distribution) archive from an installed base host. The second step (section 12.6.2) 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 12.7.1), which is a wrapper to cm-create-image.

12.6.1 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 12.6.2.
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

```bash
ssh root@basehost64 \
"tar -cz --acls --xattrs \n --exclude /etc/HOSTNAME --exclude /etc/localtime \n --exclude /proc --exclude /lost+found --exclude /sys \n --exclude /root/.ssh --exclude /var/lib/dhcpd/* \n --exclude /media/floppy --exclude /etc/motd \n --exclude /root/.bash_history --exclude /root/CHANGES \n --exclude /etc/udev/rules.d/*persistent*.rules \n --exclude /var/spool/mail/* --exclude /rhn \n --exclude /etc/sysconfig/rhn/systemid --exclude /tmp/* \n --exclude /var/spool/up2date/* --exclude /var/log/* \n --exclude /etc/sysconfig/rhn/systemid.save \n --exclude /root/mbox --exclude /var/cache/yum/* \n --exclude /etc/cron.daily/rhn-updates /" > /tmp/BASEDIST.tar.gz
```

Or alternatively, a fully expanded archive file tree can be created from `basehost64` by `rsync`ing to an existing directory (here it is `/cm/images/new-image`):

Example

```bash
rsync -av --acls --xattrs --hard-links --numeric-ids \n --exclude=/etc/HOSTNAME --exclude=/etc/localtime --exclude=/proc \n --exclude=/lost+found --exclude=/sys --exclude=/root/.ssh \n --exclude=/var/lib/dhcpd/* --exclude=/media/floppy \n --exclude=/etc/motd --exclude=/root/.bash_history \n --exclude=/root/CHANGES --exclude=/var/spool/mail/* \n --exclude=/etc/udev/rules.d/*persistent*.rules \n --exclude=/rhn --exclude=/etc/sysconfig/rhn/systemid \n --exclude=/etc/sysconfig/rhn/systemid.save --exclude=/tmp/* \n --exclude=/var/spool/up2date/* --exclude=/var/log/* \n --exclude=/root/mbox --exclude=/var/cache/yum/* \n --exclude=/etc/cron.daily/rhn-updates \n 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

```bash
[bright92->fspart]% foreach * (set rsyncxattr no)
[bright92->fspart]% list -f path:0,rsyncxattr
path (key) rsyncxattr
----------------------------- --------------------
/cm/images/default-image no
```
Having built the archive by following the examples suggested, the first step in creating the software image is now complete.

12.6.2 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 561).

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 12.2) 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 12.6.3 of this manual.

Usage Of The cm-create-image Command

The usage information for cm-create-image lists options and examples:

**Usage:** cm-create-image  <OPTIONS1> [OPTIONS2]

**OPTIONS1:**

---------
-a | --fromarchive <archive>  Create software image from archive file of supported base distribution. Supported file formats are .tar, .tgz, .tar.gz, .tbz, and .tar.bz2. The extension must match the format.

-d | --fromdir <dir path>  Create software image from existing directory that already has valid base distribution.

-h | --fromhost <hostname>  Create software image from running host

-n | --imagename <name>  Name of software image to create in cluster management daemon database.

**OPTIONS2:**

---------
-i | --imagedir <dir name> Name of directory to be created in
/cm/images.
Contents of archive file are extracted
into this directory (default: name
specified with -n).
-r | --recreate Recreate directory specified by -i or
default, if it exists.
Default behavior: directory is overwritten
-s | --skipdist Skip install of base distribution packages
-e | --excludecmrepo Do not copy default cluster manager repo
files. (Use this option when the repo
files have been already modified in the
image directory, and hence must not be
overwritten.)
-f | --forcecreate Force non-interactive mode
-u | --updateimage If image specified by -n already exists,
then it is updated, with the new parameters
-b | --basedistrepo <file> Use this as the repo configuration file
for fetching required distribution
packages (cannot be used with -e).
-c | --cmrepo <file> Use this as the repo configuration file
for fetching required cluster manager
packages (cannot be used with -e).
-m | --minimal Install only a minimal set of packages,
relevant for the cluster management
decadeon to function. Use with -s option, to
also prevent additional distribution
packages from being installed.
-w | --hwvendor Install hardware vendor specific packages.
Valid choices are: dell|cray|ciscoucs|hp|
ibm|supermicro|other
-l | --resolvconf resolv.conf to be used inside image dir,
during image creation (by default
/etc/resolv.conf from head node is used)
-x | --excludecm <list> List of CM packages to exclude (comma-
separated)
-o | --exclude-from <file> File containing the exclude patterns in the
format of the 'rsync' command. See the
'exclude/include pattern rules' in the
documentation of 'rsync' for more information.
-j | --excludedist <list> List of distribution packages to exclude
(comma-separated)
-q | --excludewvendor <list> List of hardware vendor packages to
exclude (comma-separated)
-g | --enableextrarepo <value> Argument must be the special string
'public' or path to a directory that has
the Bright DVD/ISO mounted.

EXAMPLES:
---------
1. cm-create-image -a /tmp/RHEL7.tar.gz -n rhel7-image
2. cm-create-image -a /tmp/RHEL7.tar.gz -n rhel7-image -i /cm/images/test-image
3. cm-create-image -d /cm/images/SLES11-image -n sles11-image
4. cm-create-image -h node001 -n node001-image
5. cm-create-image -a /tmp/RHEL7.tar.gz -n rhel7-image -i /cm/images/new-image -r
12.6 Creating A Custom Software Image

6. `cm-create-image -a /tmp/RHEL7.tar.gz -n rhel7-image -i /cm/images/new-image -u`
7. `cm-create-image -d /cm/images/new-image -n bio-image -s -e`
8. `cm-create-image -d /cm/images/new-image -n bio-image -s -b /tmp/rhel7-updates.repo`
9. `cm-create-image -d /cm/images/new-image -n bio-image -s -c /tmp/cm-rhel7.repo`
10. `cm-create-image -d /cm/images/new-image -n bio-image -s -m`

Explanations Of The Examples In Usage Of `cm-create-image`

Explanations of the 10 examples in 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 Bright 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 12.5.2) 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. If the destination directory already exists, the `--recreate` option can be used to recreate the existing directory. The administrator should be aware that this means removal of the content of any existing directory of the same name. Thus, in the following, the content under `/cm/images/new-image/` is deleted, and new image content is taken from the base distribution archive file, `/tmp/RHEL7.tar.gz` and then placed under `/cm/images/new-image/`. Also, the software image is given the name `rhel7-image`:
cm-create-image --fromarchive /tmp/RHEL7.tar.gz --imagename rhel7-image --imagedir /cm/images/new-image --recreate

If the --recreate option is not used, then the contents are simply overwritten. It means that old files on the destination may survive unchanged because the new source may not have filenames matching those.

6. The destination directory can also just be updated without removing the existing contents, by using the option --updateimage. It is almost the same as the “contents are simply overwritten” behavior described in example 5, but it actually works like an rsync command. Thus, in the following, the base distribution archive file, /tmp/RHEL7.tar.gz, is used to update the contents under the directory /cm/images/new-image/. The name of the image is also set to rhel7-image.

   cm-create-image --fromarchive /tmp/RHEL7.tar.gz --imagename rhel7-image --imagedir /cm/images/new-image --updateimage

7. With the default cluster manager, the head node provisions a software image based on the parent distribution to the other nodes. The software image which runs on the nodes provides a selection of distribution packages from the parent distribution. The default software image is thus a selection of Red Hat packages, if the head node uses Red Hat, or a selection of SUSE packages if the head node uses SUSE, and so on. The other packages for the software image are supplied by Bright Computing. When creating a custom software image, and if using the --skipdist flag with cm-create-image, then the cluster manager packages are added to the software image, but no parent distribution packages are added. Thus, in the following, the packages made available to cm-create-image in the directory /cm/images/new-image, are installed into the image named bio-image; however, no packages matching parent distribution packages are installed (because of the --skipdist option). Furthermore, transfer of the packages takes place only if they are newer than the files already in the bio-image image (because of the --updateimage option):

   cm-create-image --fromdir /cm/images/new-image --imagename bio-image --skipdist --updateimage

   So, only the cluster manager packages are updated to the image bio-image in the directory /cm/images/bio-image.

8. 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).

9. 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).

10. The --minimal flag tells cm-create-image to install only a limited number of Bright packages and its dependencies, which are required for the cluster management daemon to run the node. This also means that no Bright configuration RPMS will be installed, and all existing system configuration files will remain untouched. The minimal package selection list is read from /cm/local/apps/cluster-tools/config/minimal. The word “minimal” primarily refers to the cluster manager, and it is still required to use the --skipdist option explained earlier, in order to prevent additional parent distribution packages from being installed. This is because “minimal” in the context of the parent distribution can vary a lot depending on the requirements of different users, and is beyond the scope of cm-create-image.

   cm-create-image --fromdir /cm/images/new-image --imagename bio-image --skipdist --minimal
12.6 Creating A Custom Software Image

Package Selection Files In cm-create-image

Regarding explanation 7 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:

```xml
<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 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`, `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 `/var/log/cmcreateimage.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 Bright View via the clickpath Provisioning→Software images→Software Images→Edit→Settings→Path
- In cmsh within softwareimage mode, for example:

[bright92->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

12.6.3 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 12.6.2, 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 562)
- Setting up a local repository for RHEL (page 563)
- Setting up a local repository for CentOS and Bright from the cluster manager ISO for CentOS (page 563)

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:

```
[root]
name=Red Hat Enterprise Linux $releasever - $basearch - Base
baseurl=file:///root/repo1/Server
gpgcheck=0
enabled=1
```

Configuring Local Repositories For CentOS And Bright Computing 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 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:

  ```
  [bright-repo]
  name=NVIDIA Bright Cluster Manager DVD Repo
  baseurl=file:///mnt1/data/cm-rpms/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@bright92 ~]# yum clean all
[root@bright92 ~]# 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.

12.6.4 Creating A Custom Image From The Local Repository

After having created the local repositories for SLES, RHEL or CentOS/Rocky (section 12.6.3), 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.

12.7 Creating Images For Other Distributions And Architectures (Multidistro And Multiarch)

NVIDIA Bright Cluster Manager version 9.0 onwards makes it easier to mix distributions in the cluster. This ability is called **multidistro**. However, it is often also loosely called **multiOS**.

NVIDIA Bright 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 NVIDIA Bright Cluster Manager 9.1 are shown in the following table:

<table>
<thead>
<tr>
<th>Image</th>
<th>RHEL7</th>
<th>RHEL8</th>
<th>Ubuntu 18.04, 20.04</th>
<th>SLES12</th>
<th>SLES15</th>
</tr>
</thead>
<tbody>
<tr>
<td>RHEL7</td>
<td>x86-64</td>
<td>x86-64</td>
<td>x86-64</td>
<td>x86-64</td>
<td>x86-64</td>
</tr>
<tr>
<td>RHEL8</td>
<td>x86-64</td>
<td>x86-64, aarch64</td>
<td>x86-64, aarch64</td>
<td>x86-64</td>
<td>x86-64</td>
</tr>
<tr>
<td>Ubuntu 18.04</td>
<td>x86-64</td>
<td>x86-64, aarch64</td>
<td>x86-64, aarch64</td>
<td>x86-64</td>
<td>x86-64</td>
</tr>
<tr>
<td>Ubuntu 20.04</td>
<td>x86-64</td>
<td>x86-64, aarch64</td>
<td>x86-64, aarch64</td>
<td>x86-64</td>
<td>x86-64</td>
</tr>
<tr>
<td>SLES12</td>
<td>-</td>
<td>x86-64</td>
<td>-</td>
<td>x86-64</td>
<td>x86-64</td>
</tr>
<tr>
<td>SLES15</td>
<td>-</td>
<td>x86-64</td>
<td>-</td>
<td>x86-64</td>
<td>x86-64</td>
</tr>
</tbody>
</table>

For example, a head node running Ubuntu 18.04, on x86-64 or ARMv8 hardware, can support an Ubuntu 18.04, Ubuntu 20.04, and RHEL8 distribution running on x86-64 and on ARMv8 hardware for the compute nodes. In addition, that same head node supports running RHEL7 and SLES15 on the compute nodes, but only for x86-64 hardware.

To configure multiarch and multidistro, the `cm-image` tool is used.

### 12.7.1 The cm-image Tool

The `cm-image` tool is essentially a wrapper for the `cm-create-image` (section 12.6) 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
- to create a separate node-installer image as well as a separate software image
- to create a directory under `/cm/shared` for each image
- to select the architecture
- to manage packages in an image more easily

When used to enable a distribution for the first time, multiple changes are made to critical files and paths that may put the regular nodes into an unstable state. This means that all regular nodes should be rebooted after its first use.

The command options of `cm-image` are illustrated by the following modes and options tree:

```plaintext
cm-image
|---------- shell
| -h|--help
| -i|--image <image>
|---------- create
| | ---- all
| | | -a|--arch <architecture> (mandatory)
```

1. The `--bootstrap` option can only create Ubuntu 18.04 or 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 kernel 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.
Creating Images For Other Distributions And Architectures (Multidistro And Multiarch)

```
--skip-connectivity-check

----- fromfile <JSON input file>
    -h|--help

'----- cmshared
    -a|--arch <architecture>     (mandatory)
    -d|--distro <distribution>   (mandatory)
    -h|--help
    -f|--force
    -i|--image <image>
    -b|--baserepo <base repository>
    -c|--cmrepo <cluster manager repository>
    -e|--excludecm <Bright packages to exclude from installation>
    -x|--excludedist <distribution packages to exclude from installation>
    -c|--baseurl <base distribution repository URL>
    -e|--extraurl <extra repository URL>
    -s|--skip-connectivity-check
    -d|--default
    --add-only

'-------- remove
    -a|--arch <architecture>     (mandatory)
    -d|--distro <distribution>   (mandatory)
    -h|--help
    -f|--force
    -e|--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>`: aarch, x86_64
- `<distribution>`: rhel7u7, rhel7u8, rhel7u9, rhel8u0, rhel8u1, rhel8u2, rhel8u3, rhel8u4, rhel8u5, sles12sp2, sles12sp3, sles12sp4, sles15sp1, sles15sp2, ubuntu1804, ubuntu2004. centos is automatically substituted for rhel for CentOS distributions, and rocky is automatically substituted for rhel for Rocky Linux distributions.
- `<archive>`: path to a base tar file (section 12.6.1), eg: /root/basetar/data/UBUNTU2004.tar.gz
- `<directory>`: path to a filesystem, eg: /root/basetar/data/untarred/
- `<host>`: URL to a host, eg: http://10.141.255.264/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`

12.7.2 Multidistro Examples: Provisioning From CentOS 7 Head Node To Ubuntu 18.04 Regular Nodes

Using Bootstrap For Cluster With Network Access To Repositories

```
[root@bright92 ~]# module load cm-image
[root@bright92 ~]# 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@bright92 ~]# cmsh
[bright92] % device use node001 
[bright92->device[node001]] % set category default-ubuntu1804-x86_64 
[bright92->device[node001]] % commit 
[bright92->device[node001]] % Thu Jan 16 14:35:08 2020 [notice] bright92: node001 [ UP ]. restart required (category) 
[bright92->device[node001]] % reboot 
node001: Reboot in progress ...
```

Using ISO For Cluster Without Network Access

A base tar (section 12.6.1) can be used
```
[root@bright92 ~]# module load cm-image 
[root@bright92 ~]# cm-image --verbose create all -a x86_64 -d ubuntu1804 --archive /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
12.7 Creating Images For Other Distributions And Architectures (Multidistro And Multiarch) 569

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@bright92 ~]# module load cm-image
[root@bright92 ~]# cm-image --verbose create all -a x86_64 -d centos7 --archive /root/basetar/data/CENTOS7u8.tar.gz
[root@bright92 ~]# cm-create-image -a /run/CENTOS7u7.tar.gz --arch x86_64 --os centos7 -f -n default-image-centos7u7-x86_64 -i /cm/images/default-image-centos7u7-x86_64 --g public
[root@bright92 ~]# cmsh
[bright92]# device use node001
[bright92->device[node001]]% set category default-centos7-x86_64
[bright92->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_64, has been copied over from another cluster:

```
[root@bright92 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@bright92 ~]# module load cm-image
[root@bright92 ~]# cm-image create swimage -a x86_64 -d ubuntu1804 --add-only
```

12.7.3 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@bright92 ~]# mkdir /var/www/html/aarch64-iso
[root@bright92 ~]# mount -o loop /root/bright91-rhel8u2.aarch64.iso /var/www/html/aarch64-iso
```

A repository file can be created with the following content:

```
[root@bright92 ~]# 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
```
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:

```bash
[root@bright92 ~]# module load cm-image
[root@bright92 ~]# cm-image --verbose create all -a aarch64 -d rhel8 --archive \
/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`:

```bash
[root@bright92 ~]# cmsh
[bright92]% category list
Name (key) Software image Nodes
------------------------ -------------------------------------- --------
default-centos8-x86_64 default-image 1
default-rhel8-aarch64 default-image-rhel8-aarch64 1

[bright92]% softwareimage list
Name (key) Path ...
--------------------------- -------------------------------------- ...
default-image /cm/images/default-image ...
default-image-rhel8-aarch64 /cm/images/default-image-rhel8-aarch64 ...

[bright92]% 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
eaarch64 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:

```bash
[root@bright92 ~]# cmsh
[bright92]% device use arm-node001
[bright92->device[node001]]% set category default-rhel8-aarch64
[bright92->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**

```
[bright92->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....
[bright92->partition[base]->archos]% use aarch64/ubuntu1804
[bright92->partition[base]->archos[aarch64/ubuntu1804]]% set primaryimage default-image default-image-ubuntu1804-aarch64 new-image-ubuntu1804-aarch64
[bright92->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@bright92 ~]# cmsh
[bright92]% category use default-rhel8-aarch64
[bright92->category[default-rhel8-aarch64]]% set bootloaderprotocol tftp
[bright92->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@bright92 ~]# cmsh
[bright92]% softwareimage use default-rhel8-aarch64
[bright92->softwareimage[default-rhel8-aarch64]]% set kernelconsoleoutput console ttyAMA0
[bright92->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**
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:

```bash
[root@bright92 ~]# cmsh
[bright92]# device monitoringresources arm-node001 | grep a64fx
a64fx
```

The monitoring settings for IPMI via the a64fx object can be enabled as follows:

```bash
[root@bright92 ~]# cmsh
[bright92]# monitoring setup; use ipmi
[bright92->monitoring->setup[ipmi]]% set script /cm/local/apps/cmd/scripts/metrics/sample_ipmitool.py
[bright92->monitoring->setup*[ipmi*]]% executionmultiplexers
[bright92->monitoring->setup*[ipmi*]->executionmultiplexers% remove ipmi
[bright92->monitoring->setup*[ipmi*]->executionmultiplexers% add resource a64fx
[bright92->monitoring->setup*[ipmi*]->executionmultiplexers*[a64fx*]]% set resources a64fx
[bright92->monitoring->setup*[ipmi*]->executionmultiplexers*[a64fx*]]% commit
```
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 Bright 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 \texttt{cmsh} use.

13.1 A Basic Monitoring Example And Action

13.1.1 Synopsis Of Basic Monitoring Example

In section 13.1, after an overview (section 13.1.1), a minimal basic example of monitoring a process is set up (section 13.1.2) and used (section 13.1.3). 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.

For 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 can monitor user mode CPU load usage throughout the cluster, and 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
Monitoring: Monitoring Cluster Devices

High load detected and processes stopped

CPU-intensive processes started

Figure 13.1: Monitoring Basic Example: CPU-intensive Processes Started, Detected And Stopped

The administrator can set that up with an action that is triggered when a high load is detected. The action that is taken after triggering, is to stop the processes (figure 13.1).

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.

13.1.2 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:

```bash
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:

```bash
#!/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.
13.1.3 Using The Basic Monitoring Example
Now that the pieces are in place, the administrator can use Bright 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 Bright View:

- The clickpath

Monitoring → Actions → Monitoring Actions → killprocess → Clone

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 574).

After saving, the killallyesaction action becomes part of the list of monitoring actions (figure 13.2).

![Figure 13.2: Bright View Monitoring Configuration: Adding An Action](image)

Setting Up A Trigger Using CPUUser On The Head Node(s)
The clickpath

Monitoring → Triggers → Failing health checks → Clone
can be used to configure a monitoring trigger, by cloning an existing trigger. A trigger is a sample state condition that runs an action. In this case, the sample state condition may be that the metric (section 13.2.3) 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.

  Unlike %user from the top command, a jiffy interval is not a percentage.

To configure this, the trigger attributes can be modified as follows (figure 13.3):

- A name is set for the trigger. The name can be arbitrary, and killallyestrigger is used in this example.

- the trigger is made active by enabling it

- 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 the expression subwindow. In this subwindow (figure 13.4):
13.1 A Basic Monitoring Example And Action

A name is set for the expression. The name can be arbitrary, and `killallyesregex` 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 `bright92` in this example, then `bright92` 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, `GT`, 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 `killallyesregex` regular 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 Bright View, as seen in figure 13.9). 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—then the `killallyesregex` regular 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.

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 13.1, 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.

13.2 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.

13.2.1 Measurables

Measurables are measurements (sample values) that are obtained via data producers (section 13.2.10) 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 13.2.2.

- **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 13.2.3.

- **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 13.2.4.

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 NVIDIA Bright 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 edgemsite etcdfspar 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@bright92 ~]# cmsh -c "softwareimage use default-image; show"
  Parameter | Value
  ----------------------------------------
  ```
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**

```
[bright92->device]% measurables node001
```

<table>
<thead>
<tr>
<th>Type</th>
<th>Name</th>
<th>Parameter</th>
<th>Class</th>
<th>Producer</th>
</tr>
</thead>
<tbody>
<tr>
<td>Enum</td>
<td>DeviceStatus</td>
<td>Internal</td>
<td>DeviceState</td>
<td></td>
</tr>
<tr>
<td>HealthCheck</td>
<td>ManagedServicesOk</td>
<td>Internal</td>
<td>CMDaemonState</td>
<td></td>
</tr>
<tr>
<td>HealthCheck</td>
<td>defaultgateway</td>
<td>Network</td>
<td>defaultgateway</td>
<td></td>
</tr>
<tr>
<td>HealthCheck</td>
<td>diskspace</td>
<td>Disk</td>
<td>diskspace</td>
<td></td>
</tr>
<tr>
<td>HealthCheck</td>
<td>dmesg</td>
<td>OS</td>
<td>dmesg</td>
<td></td>
</tr>
</tbody>
</table>

The subsets of these measurables—enummetrics, metrics, and health checks—can be listed with the `enummetrics` (section 13.2.2), `metrics` (section 13.2.3), or `healthchecks` (section 13.2.4) command.

In Bright View, all the entities that are using health checks can be viewed via the clickpath:

```
Monitoring→All Health Checks
```

**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**

```
[bright92->monitoring]% measurable list
```

<table>
<thead>
<tr>
<th>Type</th>
<th>Name (Key)</th>
<th>Parameter</th>
<th>Class</th>
<th>Producer</th>
</tr>
</thead>
<tbody>
<tr>
<td>Enum</td>
<td>DeviceStatus</td>
<td>Internal</td>
<td>DeviceState</td>
<td></td>
</tr>
<tr>
<td>HealthCheck</td>
<td>ManagedServicesOk</td>
<td>Internal</td>
<td>CMDaemonState</td>
<td></td>
</tr>
<tr>
<td>HealthCheck</td>
<td>Mon::Storage</td>
<td>Internal/Monitoring/Storage</td>
<td>MonitoringSystem</td>
<td></td>
</tr>
<tr>
<td>HealthCheck</td>
<td>chrootprocess</td>
<td>OS</td>
<td>chrootprocess</td>
<td></td>
</tr>
<tr>
<td>HealthCheck</td>
<td>cmsh</td>
<td>Internal</td>
<td>cmsh</td>
<td></td>
</tr>
</tbody>
</table>

...
The subsets of these measurables—enummetrics, metrics, and health checks—can be listed with:
list enum (section 13.2.2), list metric (section 13.2.3), or list healthcheck (section 13.2.4).
In Bright View, the equivalent to listing the measurables can be carried out via the clickpath:

Monitoring → Measurables  (figure 13.10, section 13.4.2)

and listing the subsets of the measurables is possible using column filtering (figure 13.11, section 13.4.2).

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

[bright92->monitoring->measurable]% use devicestatus
[bright92->monitoring->measurable[DeviceStatus]]% show

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>Class</td>
<td>Internal</td>
</tr>
<tr>
<td>Consolidator</td>
<td>none</td>
</tr>
<tr>
<td>Description</td>
<td>The device status</td>
</tr>
<tr>
<td>Disabled</td>
<td>no (DeviceState)</td>
</tr>
<tr>
<td>Maximal age</td>
<td>0s (DeviceState)</td>
</tr>
<tr>
<td>Maximal samples</td>
<td>4,096 (DeviceState)</td>
</tr>
<tr>
<td>Name</td>
<td>DeviceStatus</td>
</tr>
<tr>
<td>Parameter</td>
<td>DeviceState</td>
</tr>
<tr>
<td>Revision</td>
<td></td>
</tr>
<tr>
<td>Type</td>
<td>Enum</td>
</tr>
</tbody>
</table>

13.2.2 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

[bright92->monitoring->measurable]% list enum

<table>
<thead>
<tr>
<th>Type</th>
<th>Name (key)</th>
<th>Parameter</th>
<th>Class</th>
<th>Producer</th>
</tr>
</thead>
<tbody>
<tr>
<td>Enum</td>
<td>DeviceStatus</td>
<td>Internal</td>
<td>DeviceState</td>
<td></td>
</tr>
</tbody>
</table>

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
13.2 Monitoring Concepts And Definitions

The states that the entity has been through can be viewed with a dumpmonitoringdata command (section 13.6.4):

Example

```
[bright92->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 states that the entity has been through can be viewed with a dumpmonitoringdata command (section 13.6.4):

Example

```
[bright92->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
```

13.2.3 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 13.1, 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

```
[bright92->monitoring] measurable list metric
```

```
Type Name (key) Parameter Class Producer
------- ------------------------ -------------- ------------------ -------------
Metric AlertLevel count Internal AlertLevel
Metric AlertLevel maximum Internal AlertLevel
...
In Bright View, the metrics can be viewed with the clickpath:

```
Monitoring → Measurables
```

(figure 13.10, section 13.4.2)

and then clicking on the filter widget to select Metric (figure 13.11, section 13.4.2).

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**

```
[bright92->devices]% metrics node001
```

<table>
<thead>
<tr>
<th>Type</th>
<th>Name</th>
<th>Parameter</th>
<th>Class</th>
<th>Producer</th>
</tr>
</thead>
<tbody>
<tr>
<td>Metric</td>
<td>AlertLevel</td>
<td>count</td>
<td>Internal</td>
<td>AlertLevel</td>
</tr>
<tr>
<td>Metric</td>
<td>AlertLevel</td>
<td>maximum</td>
<td>Internal</td>
<td>AlertLevel</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

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**

```
[bright92->monitoring->measurable]% use alertlevel:count
[bright92->monitoring->measurable[AlertLevel:count]]% show
```

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>Class</td>
<td>Internal</td>
</tr>
<tr>
<td>Consolidator</td>
<td>default</td>
</tr>
<tr>
<td>Cumulative</td>
<td>no</td>
</tr>
<tr>
<td>Description</td>
<td>Number of active triggers</td>
</tr>
<tr>
<td>Disabled</td>
<td>no</td>
</tr>
<tr>
<td>Maximal age</td>
<td>0s</td>
</tr>
<tr>
<td>Maximal samples</td>
<td>0</td>
</tr>
<tr>
<td>Maximum</td>
<td>0</td>
</tr>
<tr>
<td>Minimum</td>
<td>0</td>
</tr>
<tr>
<td>Name</td>
<td>AlertLevel</td>
</tr>
<tr>
<td>Parameter</td>
<td>count</td>
</tr>
<tr>
<td>Producer</td>
<td>AlertLevel</td>
</tr>
<tr>
<td>Revision</td>
<td></td>
</tr>
<tr>
<td>Type</td>
<td>Metric</td>
</tr>
</tbody>
</table>

The equivalent Bright View clickpath to edit the parameters is:

```
Monitoring → Measurables → Edit
```

### 13.2.4 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.
13.2 Monitoring Concepts And Definitions

- **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 Bright View, the health checks that can be configured for all entities can be seen with the clickpath:

```
Monitoring → Measurables  (figure 13.10, section 13.4.2)
```

and then clicking on the filter widget to select Health Check (figure 13.11, section 13.4.2). Options can be set for each health check by clicking through via the Edit button.

**All Configured Health Checks**

In Bright View, health checks that have been configured for all entities can be seen with the clickpath:

```
Monitoring → All Health Checks  (section 13.4.7)
```

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 13.2.7) 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 13.2.8) 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 13.1 this is how, when the metric measurable CPUUser crosses 50 jiffies, the killallyestigger is activated, and the killallyyes action script is run.

13.2.5 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 13.2.7) associated with it. This value is processed for the AlertLevel metric (section 13.2.8) when an action is triggered by a threshold event.

Triggers are discussed further in section 13.4.5.

13.2.6 Action

In the basic example of section 13.1, 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.

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 13.2.9).

The actions that can be run are listed from within the action submode of the monitoring mode.

Example

```
[bright92->monitoring->action]% list
<table>
<thead>
<tr>
<th>Type</th>
<th>Name (key)</th>
<th>Run on</th>
<th>Action</th>
</tr>
</thead>
<tbody>
<tr>
<td>Drain</td>
<td>Drain</td>
<td>Active</td>
<td>Drain node from all WLM</td>
</tr>
<tr>
<td>Email</td>
<td>Send e-mail</td>
<td>Active</td>
<td>Send e-mail</td>
</tr>
<tr>
<td>Event</td>
<td>Event</td>
<td>Active</td>
<td>Send an event to users with connected client</td>
</tr>
<tr>
<td>ImageUpdate</td>
<td>ImageUpdate</td>
<td>Active</td>
<td>Update the image on the node</td>
</tr>
<tr>
<td>PowerOff</td>
<td>PowerOff</td>
<td>Active</td>
<td>Power off a device</td>
</tr>
<tr>
<td>PowerOn</td>
<td>PowerOn</td>
<td>Active</td>
<td>Power on a device</td>
</tr>
<tr>
<td>PowerReset</td>
<td>PowerReset</td>
<td>Active</td>
<td>Power reset a device</td>
</tr>
<tr>
<td>Reboot</td>
<td>Reboot</td>
<td>Node</td>
<td>Reboot a node</td>
</tr>
<tr>
<td>Script</td>
<td>killallyesaction</td>
<td>Node</td>
<td>/cm/local/apps/cmd/scripts/actions/killallyes</td>
</tr>
<tr>
<td>Script</td>
<td>killprocess</td>
<td>Node</td>
<td>/cm/local/apps/cmd/scripts/actions/killprocess.pl</td>
</tr>
</tbody>
</table>
```
13.2 Monitoring Concepts And Definitions

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 Bright View equivalent is accessible via the clickpath:

Monitoring→Actions  (figure 13.17, section 13.4.4)

Configuration of monitoring actions is discussed further in section 13.4.4.

13.2.7 Severity

Severity is a positive integer value that the administrator assigns for a trigger. It takes one of these 6 suggested values:

<table>
<thead>
<tr>
<th>Value</th>
<th>Name</th>
<th>Icon</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>debug</td>
<td>📋</td>
<td>debug message</td>
</tr>
<tr>
<td>0</td>
<td>info</td>
<td>📋</td>
<td>informational message</td>
</tr>
<tr>
<td>10</td>
<td>notice</td>
<td>🔄</td>
<td>normal, but significant, condition</td>
</tr>
<tr>
<td>20</td>
<td>warning</td>
<td>🔄</td>
<td>warning conditions</td>
</tr>
<tr>
<td>30</td>
<td>error</td>
<td>🔄</td>
<td>error conditions</td>
</tr>
<tr>
<td>40</td>
<td>alert</td>
<td>🔄</td>
<td>action must be taken immediately</td>
</tr>
</tbody>
</table>

Severity levels are used in the AlertLevel metric (section 13.2.8). They can also be set by the administrator in the return values of health check scripts (section 13.2.4).

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 13.2.4).

13.2.8 AlertLevel

AlertLevel is a special metric. It is sampled and re-calculated when an event with an associated Severity (section 13.2.7) 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.

13.2.9 Flapping

Flapping, or State Flapping, is when a measurable trigger that is detecting changes, (section 13.4.5) changes 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 13.1, 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.

13.2.10 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

```bash
[bright92->monitoring->measurable] % list -f name:25,producer:15 | grep ProcStat
```

<table>
<thead>
<tr>
<th>BlockedProcesses</th>
<th>ProcStat</th>
</tr>
</thead>
<tbody>
<tr>
<td>CPUGuest</td>
<td>ProcStat</td>
</tr>
<tr>
<td>CPUIdle</td>
<td>ProcStat</td>
</tr>
<tr>
<td>CPUIrrq</td>
<td>ProcStat</td>
</tr>
<tr>
<td>CPUNice</td>
<td>ProcStat</td>
</tr>
<tr>
<td>CPUSoftIrrq</td>
<td>ProcStat</td>
</tr>
<tr>
<td>CPUSteal</td>
<td>ProcStat</td>
</tr>
<tr>
<td>CPUSystem</td>
<td>ProcStat</td>
</tr>
<tr>
<td>CPUUser</td>
<td>ProcStat</td>
</tr>
<tr>
<td>CPUWait</td>
<td>ProcStat</td>
</tr>
<tr>
<td>CtxtSwitches</td>
<td>ProcStat</td>
</tr>
<tr>
<td>Forks</td>
<td>ProcStat</td>
</tr>
<tr>
<td>Interrupts</td>
<td>ProcStat</td>
</tr>
<tr>
<td>RunningProcesses</td>
<td>ProcStat</td>
</tr>
</tbody>
</table>

Sometimes it may just be one measurable, as provided by a used data producer:

Example

```bash
[bright92->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

```bash
[bright92->monitoring->setup] % list
```

The equivalent in Bright View is via the clickpath:

Monitoring → Data Producers

The data producers configured for an entity, such as a head node bright92, can be listed with the monitoringproducers command:

Example

```bash
[bright92->device[bright92]] % monitoringproducers
```

<table>
<thead>
<tr>
<th>Type</th>
<th>Name</th>
<th>Arguments</th>
<th>Measurables</th>
<th>Node execution filters</th>
</tr>
</thead>
<tbody>
<tr>
<td>AlertLevel</td>
<td>AlertLevel</td>
<td>3 / 231</td>
<td>&lt;0 in submode&gt;</td>
<td></td>
</tr>
<tr>
<td>CMDaemonState</td>
<td>CMDaemonState</td>
<td>1 / 231</td>
<td>&lt;0 in submode&gt;</td>
<td></td>
</tr>
<tr>
<td>ClusterTotal</td>
<td>ClusterTotal</td>
<td>18 / 231</td>
<td>&lt;1 in submode&gt;</td>
<td></td>
</tr>
<tr>
<td>Collection</td>
<td>NFS</td>
<td>32 / 231</td>
<td>&lt;0 in submode&gt;</td>
<td></td>
</tr>
<tr>
<td>Collection</td>
<td>sdt</td>
<td>0 / 231</td>
<td>&lt;0 in submode&gt;</td>
<td></td>
</tr>
<tr>
<td>DeviceState</td>
<td>DeviceState</td>
<td>1 / 231</td>
<td>&lt;1 in submode&gt;</td>
<td></td>
</tr>
<tr>
<td>HealthCheckScript</td>
<td>chrootprocess</td>
<td>1 / 231</td>
<td>&lt;1 in submode&gt;</td>
<td></td>
</tr>
<tr>
<td>HealthCheckScript</td>
<td>cmsh</td>
<td>1 / 231</td>
<td>&lt;1 in submode&gt;</td>
<td></td>
</tr>
<tr>
<td>HealthCheckScript</td>
<td>defaultgateway</td>
<td>1 / 231</td>
<td>&lt;0 in submode&gt;</td>
<td></td>
</tr>
<tr>
<td>HealthCheckScript</td>
<td>diskspace</td>
<td>1 / 231</td>
<td>&lt;0 in submode&gt;</td>
<td></td>
</tr>
<tr>
<td>HealthCheckScript</td>
<td>dmesg</td>
<td>1 / 231</td>
<td>&lt;0 in submode&gt;</td>
<td></td>
</tr>
</tbody>
</table>
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 Bright View is discussed further in section 13.4.1.

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 Bright 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 13.8) 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 plot-
Monitoring: Monitoring Cluster Devices

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.

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@bright92 ~]# cmsh
[bright92]% monitoring setup
[bright92->monitoring->setup]% use mounts
[bright92->...[mounts]]% set access private
[bright92->...*[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@bright92 ~]# 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@bright92 ~]# cmsh
[bright92]% monitoring measurable
[bright92->monitoring->measurable]% use loadone
[bright92->...[LoadOne]]% get access
Public (SysInfo)
[bright92->...*[LoadOne*]]% set access private
[bright92->...*[LoadOne*]]% commit
```

---

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.
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@bright92 ~]# cmsh -c "monitoring measurable; list -f name:40,access"

<table>
<thead>
<tr>
<th>name</th>
<th>access</th>
</tr>
</thead>
<tbody>
<tr>
<td>AlertLevel</td>
<td>Public</td>
</tr>
<tr>
<td>AlertLevel</td>
<td>Public</td>
</tr>
<tr>
<td>AlertLevel</td>
<td>Public</td>
</tr>
<tr>
<td>AvgJobDuration</td>
<td>Public</td>
</tr>
<tr>
<td>BlockedProcesses</td>
<td>Public</td>
</tr>
</tbody>
</table>

13.2.11 Conceptual Overview: The Main Monitoring Interfaces Of Bright View

Bright 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 Bright View standard display (figure 13.5):

The 11 icons are described from left to right next:

1. **Toggle dark theme** option allows the display of Bright 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 Bright 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 Bright View **Monitoring** mode, which is launched by the next icon in figure 13.5. The Monitoring resource is about configuring how items are monitored and how their data values are collected, and is discussed further in section 13.4.

4. The **Monitoring** mode allows visualization of the data values collected according to the specifications of the Bright View Monitoring resource. The visualization allows graphs to be configured, and is discussed further in section 13.3.

5. The **Accounting** mode typically allows visualization of job resources used by users, although it can be used to visualize job resources used by other aggregation entities. This is helpful tracking resources consumed by users. Job accounting is discussed further in Chapter 15.
6. The Chargeback mode allows the monitoring of resources requested over a period for jobs run by selected groups (Chapter 16).

7. The Events icon allows logs of events (section 13.10.1) 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 Bright View user.

The monitoring aspects of the first two icons are discussed in greater detail in the sections indicated.

13.3 Monitoring Visualization With Bright View

The Monitoring icon in the menu bar of Bright View (item 4 in figure 13.5) 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 Bright 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 13.6). 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 13.5 and 13.6.

Visualization of monitoring graphs with Bright View is now described.

13.3.1 The Monitoring Window

If the Monitoring icon is clicked on from the menu bar of Bright View (figure 13.5), 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 13.6).
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 bright92, it can be selected from the navigation clickpath Device→bright92→CPU→CPUUser.

Sometimes, finding a measurable is easier with the Search box. Typing in CPUUser there shows all the measurables with that text (figure 13.7). The search is case-insensitive.

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
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.

13.4 Monitoring Configuration With Bright View

This section is about the configuration of monitoring for measurables, and about setting up trigger actions.

If Bright View is running in the standard Settings mode, which is the first icon in figure 13.5, page 589, then selecting Monitoring from the resources section makes the following menu items available for managing or viewing:

- Data Producers (section 13.4.1)
- Measurables (section 13.4.2)
- Consolidators (section 13.4.3)
- Actions (section 13.4.4)
- Triggers (section 13.4.5)
- Health status (section 13.4.6)
- All Health checks (section 13.4.7)
- Standalone Monitored Entities (section 13.4.8)
- PromQL Queries (section 15.3)
- Resources (section 13.5.4)
- Report Queries (section 13.12)
13.4 Monitoring Configuration With Bright View

These settings (figure 13.8) are now discussed in detail.

13.4.1 Monitoring Configuration: Data Producers

The Data Producers window lists all the data producers. Data producers are introduced in section 13.2.10.

Each data producer can have its settings edited within a subwindow. For example, the ProcStat data producer, which produces data for several measurables, including CPU User, has the settings shown in figure 13.9:
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.

Some remarks about some of the data producer settings that are seen in the subwindow of figure 13.9:

- **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 120s = 30s. 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 13.4.3.

- **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 three 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. This is typically a check as part of the workload manager process (section 7.8.2).
  - **On demand**: Data producer is only run on demand, 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 \( \text{LoadOne} > 1 \) (page 928).

### 13.4.2 Monitoring Configuration: Measurables

The Measurables window lists the available measurables (figure 13.10):
There are many measurables. It can therefore be useful to use the filtering-by-column option at the top of the columns. Each filtering option also has a search box associated with it (figure 13.11):

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 13.12):
The options shown include the sampling options: Maximal age, Maximal samples, Consolidator. The sampling options work as described for data producers (section 13.4.1).

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 onwards, 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.

13.4.3 Monitoring Configuration: Consolidators

Introduction To Consolidators
The concept of consolidators is explained using simple ascii graphics in Appendix K, while the cmsh interface to the consolidators submode is discussed in section 13.5.2.

In this current section, the Bright View interface to consolidators is discussed.

In Bright View, the Consolidators window lists all consolidator groups (figure 13.13).
Subwindows allow the component consolidators to be created or modified (figure 13.14).

There are two pre-existing consolidator groups: \texttt{default} and \texttt{none}.

\textbf{The \texttt{none} Consolidator Group}

The \texttt{none} consolidator group, has no consolidators. Using a consolidator of \texttt{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, as can occur, for example, if \texttt{Maximal Age} and \texttt{Maximal samples} (section 13.4.1) for data producers are both set to 0.

\textbf{The \texttt{default} Consolidator Group}

The \texttt{default} consolidator group consists of the consolidators \texttt{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 \texttt{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 \texttt{Maximal Age} and \texttt{Maximal sample} settings prevent data values being kept.

\textbf{Other Consolidator Group Possibilities}

Other sets of custom intervals can also be defined. For example, instead of the \texttt{default} consolidator group, (\texttt{hour, day, week}), a similar group called \texttt{decimalminutes} consolidator group, (\texttt{1min, 10min, 100min, 1000min, 10000min}) could be created with the appropriate intervals (figure 13.15):
13.4 Monitoring Configuration With Bright View

Consolidator Item Settings

Consolidator items are members of the consolidator groups. The items have settings as properties, and can be managed (figure 13.16).

For the consolidator of `hour`, within the `default` consolidators group, the clickpath to edit its properties is:

`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.
600 Monitoring: Monitoring Cluster Devices

- **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_{raw \ gone}`, which is how many seconds into the past the raw data point is removed, is given by:

\[
t_{raw \ gone} = (\text{Maximal samples})_{\text{measurable}} \times (\text{Interval})_{\text{measurable}}
\]

This value is also the default consolidation time, because the consolidated data values are normally presented from `t_{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_{consolidation}`, is then given by:

\[
t_{consolidation} = t_{raw \ gone} + Offset
\]

The monitoring visualization graphs then show consolidated data from `t_{consolidation}` seconds into the past, to further into the past.\(^2\)

### 13.4.4 Monitoring Configuration: Actions

Actions are introduced in section 13.2.6. The **Actions** window (figure 13.17) displays actions that the cluster manager provides by default, and also displays any custom actions that have been created:

![Figure 13.17: Bright View Monitoring Configuration: Actions](image)

The **killallyes** script from the basic example of section 13.1 would show up here if it has been implemented.

Actions are triggered, by triggers (section 13.4.5).

By default, the following actions exist:

---

\(^2\)For completeness: the time `t_{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_{raw \ gone}`:

\[
t_{consolidation \ gone} = (\text{Maximal samples})_{\text{consolidation}} \times (\text{Interval})_{\text{consolidation}}
\]
- **Power on**: Powers on the node
- **Power off**: Powers off the node
- **Power reset**: 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
- **Image update**: Updates the node from the software image
- **Event**: Sends an event to users connected with `cmsh` or Bright View
- **killprocess**: A script to kill a process
- **remount**: A script to remount all devices
- **test action**: 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**: the action runs on the active node only
  - **node**: the action runs on the triggering 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. Rather than defining a formal syntax, some examples are given, with explanations:

  - **November—March**: November to March. June, July and August 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 June, July, and August. 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 admins`: uses the list of users in the Administrator e-mail setting in partition[base] mode
- **killprocess, and test action**: Additional options for these are:
  - `arguments`: text that can be used by the script.
  - `script`: The location of the script on the file system.

### 13.4.5 Monitoring Configuration: Triggers

Triggers are introduced in section 13.2.5. The **Triggers** window (figure 13.18) allows actions (section 13.2.6) to be triggered based on conditions defined by the cluster administrator.

![Figure 13.18: Bright View Monitoring Configuration: Triggers](image)

**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
13.4 Monitoring Configuration With Bright View

- 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 Bright 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@bright92 ~]# cmsh
  [bright92]$ monitoring action
  [bright92->monitoring->action]% use send e-mail to administrators
  [bright92->...[send e-mail to administrators]]% append recipients user1@example.com
  [bright92->...*[send e-mail to administrators*]]% commit
  ```

  Here, email alerts would go to user1@example.com, as well as to anyone already configured in administrator e-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:
The settings can be viewed with the `show` command. TAB-completion prompting can be used to suggest possible values for the settings.

- A Bright 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 saved.

The email action can then be configured in Bright 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 13.1, seen in figures 13.3 and 13.4, 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

Clicking on the `expression` option in Trigger window of figure 13.18 opens up the expression subwindow. The expression for the trigger can then be configured by setting the entity, measurable, parameter, comparison operator, and measurable value, as shown in figure 13.19:

![Figure 13.19: Bright View Monitoring Configuration: Triggers Expression](image)

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@bright92 ~]# cmsh
[bright92]# monitoring trigger add nodedown
[bright92->monitoring->trigger*[nodedown*]]% expression
[bright92->monitoring->trigger*[nodedown*]->expression*[compare*]]% show
Parameter          Value
Name               compare
Revision           MonitoringCompareExpression
Type               MonitoringCompareExpression
Entities
Measurables
Parameters
Operator           ==
Value              FAIL
Use raw no
[bright92->monitoring->trigger*[nodedown*]->expression*[compare*]]% set value down
[bright92->monitoring->trigger*[nodedown*]->expression*[compare*]]% set operator eq
[bright92->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 36) gives a helpful warning here, so that the cluster administrator can fix the setting:

```
[bright92->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
[bright92->monitoring->trigger*[nodedown*]->expression*[compare*]]% set measurables devicestatus
[bright92->monitoring->trigger*[nodedown*]->expression*[compare*]]% ..;
[bright92->monitoring->trigger*[nodedown*]]% set enteractions send e-mail to administrators
[bright92->monitoring->trigger*[nodedown*]]% commit
```

13.4.6 Monitoring Configuration: Health status

The Health status window (figure 13.20) 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 13.2.7):
In the example shown in figure 13.20 the first entity shows a severity issue, while the other devices are fine. Details of the individual health checks per node can be viewed in a subwindow using the Show button for a node.

Clicking on the Show button for the first entity in this example opens up a subwindow (figure 13.21). For this example the issue turns out to be due to an UNKNOWN status in the ssh2node measurable.

13.4.7 Monitoring Configuration: All Health Checks
The All Health checks window shows all the running health checks for all entities. If the filtering-by-column option at the top of the column for the node is used to show the results for that node only, then the results are similar to what the Show button for a node produces in section 13.4.6, figure 13.21.

### 13.4.8 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 Bright View browser could be the standalone entity. This could have its connectivity monitored by pinging it from the head node with a custom script.

### 13.5 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 Bright View in section 13.4. Similarly to how monitoring sub-windows are accessed in Bright View, the monitoring mode of cmsh is itself 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**

```bash
[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
  standalone .................... Enter standalone entity mode
  trigger ....................... Enter trigger mode
```

For convenience, a tree of modes for monitoring submodes is shown in figure 13.23.
Sections 13.5.1–13.5.6 give examples of how objects are handled under these monitoring modes. To avoid repeating similar descriptions, section 13.5.1 is relatively detailed, and is often referred to by the other sections.

13.5.1 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 Bright View clickpath:

```
Monitoring → Actions
```
described earlier in section 13.4.4:

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
---  ------  -----  -------
```
13.5 The monitoring Mode Of cmsh

<table>
<thead>
<tr>
<th>Action</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Drain</td>
<td>Drain node from all WLM</td>
</tr>
<tr>
<td>Email</td>
<td>Send e-mail to administrators</td>
</tr>
<tr>
<td>Event</td>
<td>Send an event to users with connected client</td>
</tr>
<tr>
<td>ImageUpdate</td>
<td>Update the image on the node</td>
</tr>
<tr>
<td>PowerOff</td>
<td>Power off a device</td>
</tr>
<tr>
<td>PowerOn</td>
<td>Power on a device</td>
</tr>
<tr>
<td>PowerReset</td>
<td>Power reset a device</td>
</tr>
<tr>
<td>Reboot</td>
<td>Reboot a node</td>
</tr>
<tr>
<td>Script</td>
<td>Kill process</td>
</tr>
<tr>
<td>Script</td>
<td>Remount</td>
</tr>
<tr>
<td>Script</td>
<td>Test action</td>
</tr>
<tr>
<td>Shutdown</td>
<td>Shutdown a node</td>
</tr>
<tr>
<td>Undrain</td>
<td>Undrain node from all WLM (node accepts new WLM jobs)</td>
</tr>
</tbody>
</table>

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
```

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>Action</td>
<td>Power off a device</td>
</tr>
<tr>
<td>Allowed time</td>
<td>no</td>
</tr>
<tr>
<td>Disable</td>
<td>no</td>
</tr>
<tr>
<td>Name</td>
<td>PowerOff</td>
</tr>
<tr>
<td>Revision</td>
<td></td>
</tr>
<tr>
<td>Run on</td>
<td>Active</td>
</tr>
<tr>
<td>Type</td>
<td>PowerOff</td>
</tr>
</tbody>
</table>

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
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 13.1, in section 13.1.2, the killallyes action was cloned from a similar script using a clone option in Bright 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 13.1.2 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 13.1, in section 13.1.2. 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.

### 13.5.2 The `consolidator` Submode

Consolidators are introduced in section 13.4.3. Consolidators can be managed in `cmsh` via the `consolidator` mode, which is the equivalent of the consolidators window (section 13.4.3) in Bright 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 13.4.3:

```
[bright92->monitoring->consolidator]% list
Name (key)                      Consolidators
------------------------------ ------------------------
default                        hour, day, week
none                           <0 in submode>
```
Each consolidator entry can have its parameters accessed and adjusted. For example, the parameters can be viewed with:

**Example**

```
[bright92->monitoring->consolidator]% use default
[bright92->monitoring->consolidator[default]]% show
```

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>Consolidators</td>
<td>hour, day, week</td>
</tr>
<tr>
<td>Name</td>
<td>default</td>
</tr>
</tbody>
</table>

```
[bright92->monitoring->consolidator[default]]% consolidators
[bright92->monitoring->consolidator[default]->consolidators]% list
```

<table>
<thead>
<tr>
<th>Name (key)</th>
<th>Interval</th>
</tr>
</thead>
<tbody>
<tr>
<td>day</td>
<td>1d</td>
</tr>
<tr>
<td>hour</td>
<td>1h</td>
</tr>
<tr>
<td>week</td>
<td>1w</td>
</tr>
</tbody>
</table>

```
[bright92->monitoring->consolidator[default]->consolidators[day]]% show
```

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>Interval</td>
<td>1d</td>
</tr>
<tr>
<td>Kind</td>
<td>AVERAGE</td>
</tr>
<tr>
<td>Maximal age</td>
<td>0s</td>
</tr>
<tr>
<td>Maximal samples</td>
<td>4096</td>
</tr>
<tr>
<td>Name</td>
<td>day</td>
</tr>
<tr>
<td>Offset</td>
<td>0s</td>
</tr>
</tbody>
</table>

```
[bright92->monitoring->consolidator[default]->consolidators[day]]% set maximalsamples 8192
```

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 consolidator group can be created if needed.

A Bright View way, where a decimalminutes group is created, is discussed in the example in section 13.4.3, page 598.

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**

```
[bright92]% monitoring consolidator
[bright92->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
```

**13.5.3 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 Bright View clickpath:

```
Monitoring → Measurables
```

covered earlier in section 13.4.2.

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 13.5.4).

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**

```
[bright92->monitoring->measurable]% list

<table>
<thead>
<tr>
<th>type</th>
<th>name (key)</th>
<th>parameter</th>
<th>class</th>
<th>producer</th>
</tr>
</thead>
<tbody>
<tr>
<td>Enum</td>
<td>DeviceStatus</td>
<td></td>
<td>Internal</td>
<td>DeviceState</td>
</tr>
<tr>
<td>HealthCheck</td>
<td>ManagedServicesOk</td>
<td></td>
<td>Internal</td>
<td>CMDaemonState</td>
</tr>
<tr>
<td>HealthCheck</td>
<td>Mon::Storage</td>
<td></td>
<td>Internal/Monitoring/Storage</td>
<td>MonitoringSystem</td>
</tr>
<tr>
<td>Metric</td>
<td>nfs_v3_server_total</td>
<td>Disk</td>
<td>Internal/Monitoring/Storage</td>
<td>MonitoringSystem</td>
</tr>
<tr>
<td>Metric</td>
<td>nfs_v3_server_write</td>
<td>Disk</td>
<td>Internal/Monitoring/Storage</td>
<td>MonitoringSystem</td>
</tr>
</tbody>
</table>
```

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:
13.5 The monitoring Mode Of cmsh

- 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

[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: diskspace
Producer: diskspace
Revision: 0
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 13.5.1, 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 suggest over 200 possible objects:

Example

```
[bright92->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 ipfragfailes 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

```
[bright92->monitoring->measurable]% list | head -2 ; list metric | grep -i 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)
```

**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
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 13.5.1).

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 616) is used to set up a health check. The healthcheck can then be viewed from the measurable submode.

In the basic example of section 13.1, a trigger was set up from Bright View to check if the CPUUser metric was above 50 jiffies, 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 Bright 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 13.5.4.

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:

```bash
#!/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 13.5.4 to produce a fully configured health check.

13.5.4 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 Bright View clickpath:

Monitoring→Data Producers

covered earlier in section 13.4.1.

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

```
[bright92->monitoring->setup]# list
<table>
<thead>
<tr>
<th>Type</th>
<th>Name (key)</th>
<th>Arguments</th>
<th>Measurables</th>
<th>Node execution filters</th>
</tr>
</thead>
<tbody>
<tr>
<td>AggregateNode</td>
<td>AggregateNode</td>
<td>8 / 222</td>
<td>&lt;1 in submode&gt;</td>
<td></td>
</tr>
<tr>
<td>AlertLevel</td>
<td>AlertLevel</td>
<td>3 / 222</td>
<td>&lt;1 in submode&gt;</td>
<td></td>
</tr>
<tr>
<td>CMDaemonState</td>
<td>CMDaemonState</td>
<td>1 / 222</td>
<td>&lt;0 in submode&gt;</td>
<td></td>
</tr>
<tr>
<td>ClusterTotal</td>
<td>ClusterTotal</td>
<td>18 / 222</td>
<td>&lt;1 in submode&gt;</td>
<td></td>
</tr>
<tr>
<td>Collection</td>
<td>BigDataTools</td>
<td>0 / 222</td>
<td>&lt;2 in submode&gt;</td>
<td></td>
</tr>
</tbody>
</table>
```
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:

```
[bright92->monitoring->measurable]# list | head -2; list | grep AlertLevel
```

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**

```
[bright92->monitoring->setup]# list | head -2; list | grep mount
```

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**

```
[bright92->monitoring->setup]# show alertlevel
```

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>Automatic reinitialize</td>
<td>yes</td>
</tr>
</tbody>
</table>

---

```
```
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 13.4.1. Most of these properties are inherited by the measurables associated with the data producer, which in the AlertLevel data producer case are alertlevel:count, alertlevel:maximum, and alertlevel:sum.

The setup Submode: Deeper Submodes
One level under the setup submode of monitoring mode are 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
```
Most of the default data producers that are used by the cluster run on an active head node, and often on the regular nodes.

- **nodexecutions** to restrict data producer execution

The rogueprocess (page 952) 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**

```
[bright92->monitoring->setup[rogueprocess]]% nodeexecutions
[bright92->monitoring->setup[rogueprocess]->nodeexecutions]% add<TAB><TAB>
category lua node overlay resource type
[...tup[rogueprocess]->nodeexecutions]% add node justthese
[...tup*[rogueprocess*]->nodeexecutions*[justthese*]]% show

Parameter Value
--------------------------
Filter operation Include
Name justthese
Nodes
Revision
Type Node

[...tup*[rogueprocess*]->nodeexecutions*[justthese*]]% set nodes node001,node002
[...tup*[rogueprocess*]->nodeexecutions*[justthese*]]% show

Parameter Value
--------------------------
Filter operation Include
Name justthese
Nodes node001,node002
Revision
Type Node

[...tup*[rogueprocess*]->nodeexecutions*[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:

```
[bright92->monitoring->setup]% add procpidstat slapd
[bright92->monitoring->setup*[slapd*]]% set process slapd
[bright92->monitoring->setup*[slapd*]]% set consolidator none
[bright92->monitoring->setup*[slapd*]]% nodeexecutions
[bright92->monitoring->setup*[slapd*]->nodeexecutions*[headnodes*]]% add type headnodes
[bright92->monitoring->setup*[slapd*]->nodeexecutions*[headnodes*]]% set headnode yes
[bright92->monitoring->setup*[slapd*]->nodeexecutions*[headnodes*]]% show
[bright92->monitoring->setup*[slapd*]]% commit
```
The newly-defined `slapd` metric can now have its output displayed or plotted just like any other metric:

```
[bright92->device[bright92]]% 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**

```
[bright92->device[bright92]]% 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 615 can be set to run on the active head node, by creating an arbitrary resource called `myactive`:

**Example**

```
[bright92->monitoring->setup[cpucheck]]% nodeexecutionfilters
[...tup[cpucheck]->nodeexecutionfilters]% add resource "myactive"
[...tup[cpucheck]->nodeexecutionfilters*]
Parameter       Value
----------------- -----------------------------------------------
Filter Include
Name            myactive
Operator OR
Resources       Active
Revision
Type            Resource
```

and then setting the `Resources` parameter to `Active`: 
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 13.6.7).

**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@bright92 ~]# cmsh
[bright92]% monitoring setup executionmultiplexers dmesg
```

Running the commands: `help add`, or `help set`, can be used to show the valid syntax in this submode.

Most data producers run on a head node, but sample from the regular nodes. So, for example, the `dmesg` health check from Appendix G.2.1 can be set to sample from the regular nodes by setting it to carry out execution multiplexing to specified node entities using a node multiplexer with the arbitrary name of `nodes` as follows:

**Example**

```
[bright92->monitoring->setup[dmesg]->executionmultiplexers]% add<TAB><TAB>
category lua node overlay resource type
[bright92->monitoring->setup[dmesg]->executionmultiplexers]% add node nodes
[bright92->...[dmesg*]->executionmultiplexers*[nodes*]]% show
Parameter Value
------------------------------------------------------------------------
Filter operation Include
Name nodes
Nodes
Revision
Type Node
```

```
[bright92->...[dmesg*]->executionmultiplexers*[nodes*]]% set nodes node001,node002
[bright92->...[dmesg*]->executionmultiplexers*[nodes*]]% show
Parameter Value
```
### Filter operation
Include

<table>
<thead>
<tr>
<th>Name</th>
<th>nodes</th>
</tr>
</thead>
<tbody>
<tr>
<td>Nodes</td>
<td>node001,node002</td>
</tr>
<tr>
<td>Revision</td>
<td>Node</td>
</tr>
</tbody>
</table>

The concepts and expected behavior of node execution filters and execution multiplexers is covered in more explicit detail in Appendix L.

**Job Metrics Settings:** Job metrics settings are a submode for setting job metric collection options for the JobSampler data producer (section 14.4).

#### 13.5.5 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 Bright View clickpath:

Monitoring → Standalone Monitored Entities

covered earlier in section 13.4.8.

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.

#### 13.5.6 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 Bright View clickpath:

Monitoring → Triggers

covered earlier in section 13.4.5.

By default, there are 3 triggers:

**Example**

```
[bright92->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 13.1, a trigger to run the killallyes script was configured using Bright View.

The expression that was set for the killallyes script in the basic example using Bright View can also be set in cmsh. For example:
Example

[bright92->monitoring->trigger]\% add killallyestrigger
[bright92->monitoring->trigger*[killallyestrigger*]]% show

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>Disabled</td>
<td>no</td>
</tr>
<tr>
<td>During actions</td>
<td></td>
</tr>
<tr>
<td>Enter actions</td>
<td></td>
</tr>
<tr>
<td>Leave actions</td>
<td></td>
</tr>
<tr>
<td>Mark entity as failed</td>
<td>yes</td>
</tr>
<tr>
<td>Mark entity as unknown</td>
<td>no</td>
</tr>
<tr>
<td>Name</td>
<td>killallyestrigger</td>
</tr>
<tr>
<td>Revision</td>
<td></td>
</tr>
<tr>
<td>Severity</td>
<td>10</td>
</tr>
<tr>
<td>State flapping actions</td>
<td></td>
</tr>
<tr>
<td>State flapping count</td>
<td>5</td>
</tr>
<tr>
<td>State flapping period</td>
<td>5m</td>
</tr>
<tr>
<td>expression</td>
<td>(*, *, *) == FAIL</td>
</tr>
</tbody>
</table>

[bright92->monitoring->trigger*[killallyestrigger*]]% expression
[bright92->monitoring->trigger*[killallyestrigger*]->expression[]]% show

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>Entities</td>
<td></td>
</tr>
<tr>
<td>Measurables</td>
<td></td>
</tr>
<tr>
<td>Name</td>
<td></td>
</tr>
<tr>
<td>Operator</td>
<td>EQ</td>
</tr>
<tr>
<td>Parameters</td>
<td></td>
</tr>
<tr>
<td>Revision</td>
<td></td>
</tr>
<tr>
<td>Use raw</td>
<td>no</td>
</tr>
<tr>
<td>Value</td>
<td>FAIL</td>
</tr>
</tbody>
</table>

[bright92->monitoring->trigger*[killallyestrigger*]->expression[]]% set entities bright92
[bright92->monitoring->trigger*[killallyestrigger*]->expression*[[]]% set measurables CPUUser
[bright92->monitoring->trigger*[killallyestrigger*]->expression*[[]]% set operator GT
[bright92->monitoring->trigger*[killallyestrigger*]->expression*[[]]% set value 50
[bright92->monitoring->trigger*[killallyestrigger*]->expression*[[]]% commit
[bright92->monitoring->trigger*[killallyestrigger*]->expression*[[]]% set name killallyesregex
Field Message

actions Warning: No actions were set
================================================================================
[bright92->monitoring->trigger[killallyestrigger]->expression[killallyesregex]]% exit
[bright92->monitoring->trigger[killallyestrigger]->expression[]]% exit
[bright92->monitoring->trigger[killallyestrigger]]% set enteractions killallyesname
[bright92->monitoring->trigger*[killallyestrigger*]]% commit
[bright92->monitoring->trigger*[killallyestrigger*]]%

The expression format is shown in cmsh as:

\((<entity>, <measurable>, <parameter>) <comparison operator> <value>\)

Here:

- an entity, as described in section 13.2.1, 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

...[killallyestri...ression[]]]% set entities "bright92|node001|comp...image"

In the preceding example, the entity compute could be a category, and the entity gpuimage could be a software image.

- a measurable (section 13.2.1) 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 used 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):

\((!\text{resource=category:data|category:gpu|category:hpc, FreeSpace, sd[a-z]) < 10MB}\)

The unusual syntax in the preceding example is liable to change in future versions.

At the end of section 13.5.3 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 13.1. In this section the task is continued and completed as follows:

```
[bright92->monitoring->trigger]% expression killallyestrigger

[...trigger[killallyestrigger]->expression[killallyesregex]]% get measurables
CPUUser

[...trigger[killallyestrigger]->expression[killallyesregex]]% set measurables cpucheck

[...trigger*[killallyestrigger*]->expression*[killallyesregex*]]% commit
```

### 13.6 Obtaining Monitoring Data Values

The monitoring data values that are logged by devices can be used to generate graphs using the methods in section 13.3. 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.

#### 13.6.1 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 13.5.3). 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**

```
[bright92->device]% enummetrics node001

<table>
<thead>
<tr>
<th>Type</th>
<th>Name</th>
<th>Parameter</th>
<th>Class</th>
<th>Producer</th>
</tr>
</thead>
<tbody>
<tr>
<td>Enum</td>
<td>DeviceStatus</td>
<td>Internal</td>
<td>DeviceState</td>
<td>DeviceState</td>
</tr>
</tbody>
</table>

[bright92->device]% use bright92

[bright92->device]% measurables

<table>
<thead>
<tr>
<th>Type</th>
<th>Name</th>
<th>Parameter</th>
<th>Class</th>
<th>Producer</th>
</tr>
</thead>
<tbody>
<tr>
<td>Enum</td>
<td>DeviceStatus</td>
<td>Internal</td>
<td>DeviceState</td>
<td>DeviceState</td>
</tr>
<tr>
<td>HealthCheck</td>
<td>ManagedServicesOk</td>
<td>Internal</td>
<td>CMDaemonState</td>
<td></td>
</tr>
</tbody>
</table>
```

...
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 13.5.3.

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

```
[bright92->device]% partition use base
[bright92->partition[base]]% measurables
```

<table>
<thead>
<tr>
<th>Type</th>
<th>Name</th>
<th>Parameter</th>
<th>Class</th>
<th>Producer</th>
</tr>
</thead>
<tbody>
<tr>
<td>Metric</td>
<td>CoresTotal</td>
<td>Total</td>
<td>ClusterTotal</td>
<td></td>
</tr>
<tr>
<td>Metric</td>
<td>CoresUp</td>
<td>Total</td>
<td>ClusterTotal</td>
<td></td>
</tr>
<tr>
<td>Metric</td>
<td>DevicesClosed</td>
<td>Total</td>
<td>ClusterTotal</td>
<td></td>
</tr>
<tr>
<td>Metric</td>
<td>DevicesDown</td>
<td>Total</td>
<td>ClusterTotal</td>
<td></td>
</tr>
</tbody>
</table>

The values for metric samples and health checks can be obtained from within device mode in various ways, and are explained next.

13.6.2 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

```
[bright92->device]% use bright92
[bright92->device[bright92]]% samplenow
```

<table>
<thead>
<tr>
<th>Measurable</th>
<th>Parameter</th>
<th>Type</th>
<th>Value</th>
<th>Age</th>
<th>Info</th>
</tr>
</thead>
<tbody>
<tr>
<td>AlertLevel</td>
<td>count</td>
<td>Internal</td>
<td>0</td>
<td>2.01s</td>
<td></td>
</tr>
<tr>
<td>AlertLevel</td>
<td>maximum</td>
<td>Internal</td>
<td>0</td>
<td>2.01s</td>
<td></td>
</tr>
<tr>
<td>AlertLevel</td>
<td>sum</td>
<td>Internal</td>
<td>0</td>
<td>2.01s</td>
<td></td>
</tr>
<tr>
<td>BlockedProcesses</td>
<td>OS</td>
<td>0 processes</td>
<td>0</td>
<td>2.01s</td>
<td></td>
</tr>
<tr>
<td>BufferMemory</td>
<td>Memory</td>
<td>847 KiB</td>
<td>0</td>
<td>2.01s</td>
<td></td>
</tr>
</tbody>
</table>

The entity used can also be in other modes that have measurables, such as the base partition (output truncated):

Example
13.6 Obtaining Monitoring Data Values

```
[bright92->device] % partition use base
[bright92->partition[base]] % samplenow
```

<table>
<thead>
<tr>
<th>Measurable</th>
<th>Parameter</th>
<th>Type</th>
<th>Value</th>
<th>Age</th>
<th>Info</th>
</tr>
</thead>
<tbody>
<tr>
<td>CoresTotal</td>
<td>Total</td>
<td>24</td>
<td>0.001s</td>
<td></td>
<td></td>
</tr>
<tr>
<td>CoresUp</td>
<td>Total</td>
<td>24</td>
<td>0.001s</td>
<td></td>
<td></td>
</tr>
<tr>
<td>DevicesClosed</td>
<td>Total</td>
<td>0</td>
<td>0.001s</td>
<td></td>
<td></td>
</tr>
<tr>
<td>DevicesDown</td>
<td>Total</td>
<td>0</td>
<td>0.001s</td>
<td></td>
<td></td>
</tr>
<tr>
<td>DevicesTotal</td>
<td>Total</td>
<td>0</td>
<td>0.001s</td>
<td></td>
<td></td>
</tr>
<tr>
<td>DevicesUp</td>
<td>Total</td>
<td>0</td>
<td>0.001s</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

...

**The -n|--nodes Option**

The -n option is used to sample specified nodes or node ranges:

**Example**

```
[bright92->partition[base]] % device
[bright92->device] % samplenow -n node001..node002 loadone
```

<table>
<thead>
<tr>
<th>Entity</th>
<th>Measurable</th>
<th>Parameter</th>
<th>Type</th>
<th>Value</th>
<th>Age</th>
<th>Info</th>
</tr>
</thead>
<tbody>
<tr>
<td>node001</td>
<td>LoadOne</td>
<td>OS</td>
<td>0.04</td>
<td>0.08s</td>
<td></td>
<td></td>
</tr>
<tr>
<td>node002</td>
<td>LoadOne</td>
<td>OS</td>
<td>0</td>
<td>0.077s</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

**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**

```
[bright92->device] % samplenow --metrics loadone loadfifteen --n node001,node002
```

<table>
<thead>
<tr>
<th>Entity</th>
<th>Measurable</th>
<th>Parameter</th>
<th>Type</th>
<th>Value</th>
<th>Age</th>
<th>Info</th>
</tr>
</thead>
<tbody>
<tr>
<td>node001</td>
<td>LoadOne</td>
<td>OS</td>
<td>0.04</td>
<td>0.08s</td>
<td></td>
<td></td>
</tr>
<tr>
<td>node002</td>
<td>LoadOne</td>
<td>OS</td>
<td>0</td>
<td>0.077s</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

```
[bright92->device] % samplenow --checks -n node001..node002
```

<table>
<thead>
<tr>
<th>Entity</th>
<th>Measurable</th>
<th>Parameter</th>
<th>Type</th>
<th>Value</th>
<th>Age</th>
<th>Info</th>
</tr>
</thead>
<tbody>
<tr>
<td>node001</td>
<td>ManagedServicesOk</td>
<td>Internal</td>
<td>PASS</td>
<td>0.177s</td>
<td></td>
<td></td>
</tr>
<tr>
<td>node001</td>
<td>defaultgateway</td>
<td>Network</td>
<td>PASS</td>
<td>0.145s</td>
<td></td>
<td></td>
</tr>
<tr>
<td>node001</td>
<td>diskspace</td>
<td>Disk</td>
<td>PASS</td>
<td>0.16s</td>
<td></td>
<td></td>
</tr>
<tr>
<td>node001</td>
<td>dmesg</td>
<td>OS</td>
<td>PASS</td>
<td>0.177s</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

```
[bright92->device] % samplenow --checks diskspace -n node001..node002
```

<table>
<thead>
<tr>
<th>Entity</th>
<th>Measurable</th>
<th>Parameter</th>
<th>Type</th>
<th>Value</th>
<th>Age</th>
<th>Info</th>
</tr>
</thead>
<tbody>
<tr>
<td>node001</td>
<td>diskspace</td>
<td>Disk</td>
<td>PASS</td>
<td>0.095s</td>
<td></td>
<td></td>
</tr>
<tr>
<td>node002</td>
<td>diskspace</td>
<td>Disk</td>
<td>PASS</td>
<td>0.097s</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

**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**
Monitoring: Monitoring Cluster Devices

```
[bright92->device[node001]]% samplenow mtp

Measurable  Type  Value  Age  Info
-----------  ------  ------  ------  -------
  mtp        Internal  PASS  0.51s
```

```
[bright92->device[node001]]% samplenow --debug mtp

Measurable  Type  Value  Age  Info
-----------  ------  ------  ------  -------
  mtp        Internal  PASS  0.524s  command: "ps -e"
```

```
[bright92->device[node001]]% samplenow --debug -v mtp

Measurable  Type  Value  Age  Info
-----------  ------  ------  ------  -------
  mtp        Internal  PASS  0.543s  command: "ps -e"
```

```
tpd 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
```

```
[bright92->device[node001]]% !service ntpd stop
Redirecting to /bin/systemctl stop ntpd.service

[bright92->device[node001]]% samplenow --debug -v mtp

measurable  Type  Value  Age  Info
---------  ------  ------  ------  -------
  mtp        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**

```
[bright92->device]% samplenow -s UP

Entity  Measurable  Parameter  Type  Value  Age  Info
---------  ---------  --------  ------  ------  ------
  bright92  AlertLevel  count  Internal  0  4.67s
  bright92  AlertLevel  maximum Internal  0  4.67s
  bright92  AlertLevel  sum  Internal  0  4.67s
  bright92  BlockedProcesses  0S  0 processes  4.67s
  bright92  BufferMemory  Memory  847 KiB  4.67s
  bright92  BytesRecv  eth0  Network  357 MiB  4.67s
  bright92  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**

```bash
[bright92->device]% samplenow -s UP -n node001
```

<table>
<thead>
<tr>
<th>Measurable</th>
<th>Parameter</th>
<th>Type</th>
<th>Value</th>
<th>Age</th>
<th>Info</th>
</tr>
</thead>
<tbody>
<tr>
<td>AlertLevel</td>
<td>count</td>
<td>Internal</td>
<td>0</td>
<td>0.081s</td>
<td></td>
</tr>
<tr>
<td>AlertLevel</td>
<td>maximum</td>
<td>Internal</td>
<td>0</td>
<td>0.081s</td>
<td></td>
</tr>
<tr>
<td>AlertLevel</td>
<td>sum</td>
<td>Internal</td>
<td>0</td>
<td>0.081s</td>
<td></td>
</tr>
</tbody>
</table>

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 13.6.3) 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 13.6.4) displays monitoring data gathered over a period of time in a variety of formats.

### 13.6.3 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 627) 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 627) 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 626) 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**

```bash
[bright92->device]% use node001
[bright92->device[node001]]% latestmetricdata
```

<table>
<thead>
<tr>
<th>Measurable</th>
<th>Parameter</th>
<th>Type</th>
<th>Value</th>
<th>Age</th>
<th>Info</th>
</tr>
</thead>
<tbody>
<tr>
<td>AlertLevel</td>
<td>count</td>
<td>Internal</td>
<td>0</td>
<td>1m 12s</td>
<td>FAIL schedulers</td>
</tr>
<tr>
<td>AlertLevel</td>
<td>maximum</td>
<td>Internal</td>
<td>0</td>
<td>1m 12s</td>
<td>FAIL schedulers</td>
</tr>
<tr>
<td>AlertLevel</td>
<td>sum</td>
<td>Internal</td>
<td>0</td>
<td>1m 12s</td>
<td></td>
</tr>
</tbody>
</table>
### BlockedProcesses
- **OS**: 0 processes 1m 12s

### BufferMemory
- **Memory**: 847 KiB 1m 12s

### BytesRecv
- **eth0**: Network 311.611 B/s 1m 12s
- **eth1**: Network 0 B/s 1m 12s

### BytesSent
- **eth0**: Network 349.953 B/s 1m 12s
- **eth1**: Network 0 B/s 1m 12s

### CPU
- **CPU**: 0 Jiffies 1m 12s

---

Valid entity grouping options and other options can be seen in the help text for the `latestmetricdata` and `latesthealthdata` commands.

**Example**

```
[bright92->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**

```
[bright92->device]% latestmetricdata -g mynodegroup

<table>
<thead>
<tr>
<th>Entity</th>
<th>Measurable</th>
<th>Parameter</th>
<th>Type</th>
<th>Value</th>
<th>Age</th>
<th>State</th>
<th>Info</th>
</tr>
</thead>
<tbody>
<tr>
<td>node001</td>
<td>AlertLevel</td>
<td>count</td>
<td>Internal</td>
<td>0</td>
<td>15.4s</td>
<td></td>
<td></td>
</tr>
<tr>
<td>node001</td>
<td>AlertLevel</td>
<td>maximum</td>
<td>Internal</td>
<td>0</td>
<td>15.4s</td>
<td></td>
<td></td>
</tr>
<tr>
<td>node001</td>
<td>AlertLevel</td>
<td>sum</td>
<td>Internal</td>
<td>0</td>
<td>15.4s</td>
<td></td>
<td></td>
</tr>
<tr>
<td>node001</td>
<td>BlockedProcesses</td>
<td>OS</td>
<td>0 processes</td>
<td>1m 41s</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>node001</td>
<td>BufferMemory</td>
<td>Memory</td>
<td>27.3 KiB</td>
<td>41s</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>node001</td>
<td>BytesRecv</td>
<td>ens3</td>
<td>Network</td>
<td>622.722 B/s</td>
<td>56s</td>
<td></td>
<td></td>
</tr>
<tr>
<td>node001</td>
<td>BytesSent</td>
<td>ens3</td>
<td>Network</td>
<td>580.088 B/s</td>
<td>56s</td>
<td></td>
<td></td>
</tr>
<tr>
<td>node002</td>
<td>AlertLevel</td>
<td>count</td>
<td>Internal</td>
<td>0</td>
<td>15.4s</td>
<td></td>
<td></td>
</tr>
<tr>
<td>node002</td>
<td>AlertLevel</td>
<td>maximum</td>
<td>Internal</td>
<td>0</td>
<td>15.4s</td>
<td></td>
<td></td>
</tr>
<tr>
<td>node002</td>
<td>AlertLevel</td>
<td>sum</td>
<td>Internal</td>
<td>0</td>
<td>15.4s</td>
<td></td>
<td></td>
</tr>
<tr>
<td>node002</td>
<td>BlockedProcesses</td>
<td>OS</td>
<td>0 processes</td>
<td>1m 20s</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>node002</td>
<td>BufferMemory</td>
<td>Memory</td>
<td>39 KiB</td>
<td>2m 20s</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>node002</td>
<td>BytesRecv</td>
<td>ens3</td>
<td>Network</td>
<td>696.364 B/s</td>
<td>35.6s</td>
<td></td>
<td></td>
</tr>
<tr>
<td>node002</td>
<td>BytesSent</td>
<td>ens3</td>
<td>Network</td>
<td>574.08 B/s</td>
<td>35.6s</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
```
and the entities that are displayed, for example, in nodegroup mode:

Example

[bright92->nodegroup]% latestmetricdata mynodegroup

<table>
<thead>
<tr>
<th>Measurable Parameter</th>
<th>Type</th>
<th>Value</th>
<th>Age</th>
<th>State</th>
<th>Info</th>
</tr>
</thead>
<tbody>
<tr>
<td>CoresTotal</td>
<td>Total</td>
<td>4</td>
<td>31.8s</td>
<td></td>
<td></td>
</tr>
<tr>
<td>CoresUp</td>
<td>Total</td>
<td>4</td>
<td>31.8s</td>
<td></td>
<td></td>
</tr>
<tr>
<td>FPGAsTotal</td>
<td>Total</td>
<td>0</td>
<td>31.8s</td>
<td></td>
<td></td>
</tr>
<tr>
<td>FPGAsUp</td>
<td>Total</td>
<td>0</td>
<td>31.8s</td>
<td></td>
<td></td>
</tr>
<tr>
<td>GPUsTotal</td>
<td>Total</td>
<td>0</td>
<td>31.8s</td>
<td></td>
<td></td>
</tr>
<tr>
<td>GPUsUp</td>
<td>Total</td>
<td>0</td>
<td>31.8s</td>
<td></td>
<td></td>
</tr>
<tr>
<td>NodesClosed</td>
<td>Total</td>
<td>0</td>
<td>31.8s</td>
<td></td>
<td></td>
</tr>
<tr>
<td>NodesDown</td>
<td>Total</td>
<td>0</td>
<td>31.8s</td>
<td></td>
<td></td>
</tr>
<tr>
<td>NodesTotal</td>
<td>Total</td>
<td>2</td>
<td>31.8s</td>
<td></td>
<td></td>
</tr>
<tr>
<td>NodesUp</td>
<td>Total</td>
<td>2</td>
<td>31.8s</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

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

[bright92->device]% latestmonitoringcounters node001

<table>
<thead>
<tr>
<th>Measurable Parameter</th>
<th>Type</th>
<th>Value</th>
<th>Age</th>
<th>Info</th>
</tr>
</thead>
<tbody>
<tr>
<td>BytesRecv</td>
<td>eth0</td>
<td>Network</td>
<td>286 MiB</td>
<td>11.7s</td>
</tr>
<tr>
<td>BytesRecv</td>
<td>eth1</td>
<td>Network</td>
<td>0 B</td>
<td>11.7s</td>
</tr>
<tr>
<td>BytesSent</td>
<td>eth0</td>
<td>Network</td>
<td>217 MiB</td>
<td>11.7s</td>
</tr>
<tr>
<td>BytesSent</td>
<td>eth1</td>
<td>Network</td>
<td>0 B</td>
<td>11.7s</td>
</tr>
<tr>
<td>CPUGuest</td>
<td>CPU</td>
<td>0 Jiffies</td>
<td>11.7s</td>
<td></td>
</tr>
<tr>
<td>CPUIdle</td>
<td>CPU</td>
<td>60.1 Mjiffies</td>
<td>11.7s</td>
<td></td>
</tr>
<tr>
<td>CPUIrq</td>
<td>CPU</td>
<td>0 Jiffies</td>
<td>11.7s</td>
<td></td>
</tr>
<tr>
<td>CPUNice</td>
<td>CPU</td>
<td>66 Jiffies</td>
<td>11.7s</td>
<td></td>
</tr>
</tbody>
</table>

The reader can compare the preceding example output against the example output of the latestmetricdata command (page 629) to become familiar with the meaning of cumulative output.

13.6.4 Data Values Over A Period—The dumpmonitoringdata Command

The dumpmonitoringdata command displays monitoring data values over a specified period. This is for an entity, such as:

- a node in device mode
- the base partition in partition mode
- an image in softwareimage mode
- a job in the jobs submode. The jobs submode is under the path cmsh→wlm[<workload manager>]→jobs, and using dumpmonitoringdata with it is covered on page 667.
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 13.6.1).

- 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:

<table>
<thead>
<tr>
<th>Unit</th>
<th><code>&lt;start-time&gt;</code></th>
<th><code>&lt;end-time&gt;</code></th>
</tr>
</thead>
<tbody>
<tr>
<td>seconds</td>
<td>-&lt;number&gt;s</td>
<td>+&lt;number&gt;s</td>
</tr>
<tr>
<td>minutes</td>
<td>-&lt;number&gt;m</td>
<td>+&lt;number&gt;m</td>
</tr>
<tr>
<td>hours</td>
<td>-&lt;number&gt;h</td>
<td>+&lt;number&gt;h</td>
</tr>
<tr>
<td>days</td>
<td>-&lt;number&gt;d</td>
<td>+&lt;number&gt;d</td>
</tr>
</tbody>
</table>

**The options:** The options applied to the samples are specified as follows:
13.6 Obtaining Monitoring Data Values

<table>
<thead>
<tr>
<th>Option</th>
<th>Argument(s)</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>-v, --verbose</td>
<td></td>
<td>show the rest of the line on a new line instead of cutting it off</td>
</tr>
<tr>
<td>-i, --intervals</td>
<td>&lt;number&gt;</td>
<td>number of samples to show</td>
</tr>
<tr>
<td>-u, --unix</td>
<td></td>
<td>use a unix timestamp instead of using the default date format</td>
</tr>
<tr>
<td>-d, --delimiter</td>
<td>&quot;&lt;string&gt;&quot;</td>
<td>set the delimiter to a character</td>
</tr>
<tr>
<td></td>
<td>--sum</td>
<td>sum over specified entities</td>
</tr>
<tr>
<td></td>
<td>--max</td>
<td>maximum over specified entities</td>
</tr>
<tr>
<td></td>
<td>--min</td>
<td>minimum over specified entities</td>
</tr>
<tr>
<td></td>
<td>--avg</td>
<td>average over specified entities</td>
</tr>
<tr>
<td></td>
<td>--raw</td>
<td>show the metric value without units</td>
</tr>
<tr>
<td></td>
<td>--human</td>
<td>show the metric value with units (default)</td>
</tr>
<tr>
<td></td>
<td>--consolidationinterval</td>
<td>retrieve data from the consolidator with specified interval</td>
</tr>
<tr>
<td></td>
<td>--consolidationoffset</td>
<td>retrieve data from the consolidator with specified (interval, offset)</td>
</tr>
<tr>
<td></td>
<td>--timeaverage</td>
<td>calculate the average for the entire interval for specified devices</td>
</tr>
<tr>
<td></td>
<td>--delta</td>
<td>display change relative to previous value</td>
</tr>
<tr>
<td></td>
<td>--clip</td>
<td>clip data samples to the requested interval</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td>The following options are valid only for device mode:</td>
</tr>
<tr>
<td>-n, --nodes</td>
<td>&lt;list&gt;</td>
<td>for list of nodes</td>
</tr>
<tr>
<td>-g, --groups</td>
<td>&lt;list&gt;</td>
<td>for list of groups</td>
</tr>
<tr>
<td>-c, --categories</td>
<td>&lt;list&gt;</td>
<td>for list of categories</td>
</tr>
<tr>
<td>-r, --racks</td>
<td>&lt;list&gt;</td>
<td>for list of racks</td>
</tr>
<tr>
<td>-h, --chassis</td>
<td>&lt;list&gt;</td>
<td>for list of chassis</td>
</tr>
<tr>
<td>-e, --overlay</td>
<td>&lt;list&gt;</td>
<td>Include all nodes in list of overlays</td>
</tr>
<tr>
<td></td>
<td>--union</td>
<td>calculate the union of specified devices</td>
</tr>
<tr>
<td></td>
<td>--intersection</td>
<td>calculate the intersection of the specified devices</td>
</tr>
<tr>
<td>-l, --role</td>
<td>&lt;role&gt;</td>
<td>Filter all nodes in role</td>
</tr>
<tr>
<td>-s, --status</td>
<td>&lt;state&gt;</td>
<td>for nodes in state UP, OPENING, DOWN, and so on</td>
</tr>
</tbody>
</table>

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

```
[bright92->device[node001]]% dumpmonitoringdata 18:00:00 18:02:00 loadone
```

<table>
<thead>
<tr>
<th>Timestamp</th>
<th>Value</th>
<th>Info</th>
</tr>
</thead>
<tbody>
<tr>
<td>2017/08/30 17:58:00</td>
<td>0.02</td>
<td></td>
</tr>
<tr>
<td>2017/08/30 18:00:00</td>
<td>0.01</td>
<td></td>
</tr>
<tr>
<td>2017/08/30 18:02:00</td>
<td>0.02</td>
<td></td>
</tr>
</tbody>
</table>

Double quotes are needed for times with a YY/MM/DD specification:
Example

```
[bright92->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

```
[bright92->device[node001]]% date -d "Aug 30 18:00:00 2017" +%s
1504108800
[bright92->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

```
[bright92->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

```
[bright92->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:

Example

```
[bright92->device]% dumpmonitoringdata --n node001..node002 -5m now cpuidle
entity | timestamp | value | info
------------------------------- ---------- ----------
node001 2017/07/20 20:14:00 99.8258 jiffies
```
13.6 Obtaining Monitoring Data Values

node001  2017/07/20 20:16:00  99.8233 Jiffies
node001  2017/07/20 20:18:00  99.8192 Jiffies
node001  2017/07/20 20:20:00  99.8475 Jiffies
node002  2017/07/20 20:14:00  99.7917 Jiffies
node002  2017/07/20 20:16:00  99.8083 Jiffies
node002  2017/07/20 20:18:00  99.7992 Jiffies
node002  2017/07/20 20:20:00  99.815  Jiffies

[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

[device]\% dumpmonitoringdata -2h now -i 3 loadone -n node00[1-2] --sum

<table>
<thead>
<tr>
<th>Timestamp</th>
<th>Value</th>
<th>Info</th>
</tr>
</thead>
<tbody>
<tr>
<td>2017/07/20 18:30:27</td>
<td>0.0292462</td>
<td></td>
</tr>
<tr>
<td>2017/07/20 19:30:27</td>
<td>0</td>
<td></td>
</tr>
<tr>
<td>2017/07/20 20:30:27</td>
<td>no data</td>
<td></td>
</tr>
</tbody>
</table>

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

[device]\% dumpmonitoringdata -2h now -i 3 loadone -n node00[1-2]

<table>
<thead>
<tr>
<th>Entity</th>
<th>Timestamp</th>
<th>Value</th>
<th>Info</th>
</tr>
</thead>
<tbody>
<tr>
<td>node001</td>
<td>2017/07/20 18:30:27</td>
<td>0</td>
<td></td>
</tr>
<tr>
<td>node001</td>
<td>2017/07/20 19:30:27</td>
<td>0</td>
<td></td>
</tr>
<tr>
<td>node001</td>
<td>2017/07/20 20:30:27</td>
<td>no data</td>
<td></td>
</tr>
<tr>
<td>node002</td>
<td>2017/07/20 18:30:27</td>
<td>0.0292462</td>
<td></td>
</tr>
<tr>
<td>node002</td>
<td>2017/07/20 19:30:27</td>
<td>0</td>
<td></td>
</tr>
<tr>
<td>node002</td>
<td>2017/07/20 20:30:27</td>
<td>no data</td>
<td></td>
</tr>
</tbody>
</table>

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

[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

<table>
<thead>
<tr>
<th>Entity</th>
<th>Timestamp</th>
<th>Value</th>
<th>Info</th>
</tr>
</thead>
<tbody>
<tr>
<td>2020/11/03 09:56:05</td>
<td>0.010954</td>
<td></td>
<td></td>
</tr>
<tr>
<td>2020/11/03 10:56:05</td>
<td>0.000000</td>
<td></td>
<td></td>
</tr>
<tr>
<td>2020/11/03 11:56:05</td>
<td>nan</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
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**

```
[bright92->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

<table>
<thead>
<tr>
<th>Entity</th>
<th>Timestamp</th>
<th>Value</th>
<th>Info</th>
</tr>
</thead>
<tbody>
<tr>
<td>node001</td>
<td>2020/11/03 09:56:05</td>
<td>0.0109537</td>
<td></td>
</tr>
<tr>
<td>node001</td>
<td>2020/11/03 10:56:05</td>
<td>0</td>
<td></td>
</tr>
<tr>
<td>node001</td>
<td>2020/11/03 11:56:05</td>
<td>no data</td>
<td></td>
</tr>
<tr>
<td>node002</td>
<td>2020/11/03 09:56:05</td>
<td>0</td>
<td></td>
</tr>
<tr>
<td>node002</td>
<td>2020/11/03 10:56:05</td>
<td>0</td>
<td></td>
</tr>
<tr>
<td>node002</td>
<td>2020/11/03 11:56:05</td>
<td>no data</td>
<td></td>
</tr>
</tbody>
</table>
```

Similarly, for the preceding table, if the `--min` option is used instead, then the result would be:

**Example**

```
[bright92->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

<table>
<thead>
<tr>
<th>Entity</th>
<th>Timestamp</th>
<th>Value</th>
<th>Info</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>2020/11/03 09:56:05</td>
<td>0.000000</td>
<td></td>
</tr>
<tr>
<td></td>
<td>2020/11/03 10:56:05</td>
<td>0.000000</td>
<td></td>
</tr>
<tr>
<td></td>
<td>2020/11/03 11:56:05</td>
<td>nan</td>
<td></td>
</tr>
</tbody>
</table>
```

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**

```
[bright92->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

<table>
<thead>
<tr>
<th>Timestamp</th>
<th>Value</th>
<th>Info</th>
</tr>
</thead>
<tbody>
<tr>
<td>2019/03/12 14:18:15</td>
<td>201.942 B/s</td>
<td></td>
</tr>
<tr>
<td>2019/03/12 14:20:15</td>
<td>217.883 B/s</td>
<td></td>
</tr>
<tr>
<td>2019/03/12 14:22:15</td>
<td>233.058 B/s</td>
<td></td>
</tr>
<tr>
<td>2019/03/12 14:23:15</td>
<td>235.831 B/s</td>
<td></td>
</tr>
</tbody>
</table>
```

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 \texttt{-s|--status} option selects only for nodes with the specified state. A state is one of the values output by the \texttt{cmsh} command \texttt{ds} or \texttt{device status}. It is also one of the values returned by the enummetric \texttt{DeviceStatus} (section 13.2.2).

\textbf{Example}

\begin{verbatim}
[bright92->device]% dumpmonitoringdata -2m now loadone -s up

Entity | Timestamp   | Value  | Info
-------|-------------|--------|------
bright92 | 2017/07/21 15:00:00 | 0.35   |     
bright92 | 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   |     
\end{verbatim}

The argument to \texttt{-s|--status} can be specified with simple regexes, which are case insensitive. For example, \texttt{inst.*} covers the states \texttt{installing}, \texttt{installer\_failed}, \texttt{installer\_rebooting}, \texttt{installer\_callinginit}, \texttt{installer\_unreachable}, \texttt{installer\_burning}.

Displaying deltas:  The \texttt{--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.

\textbf{Example}

\begin{verbatim}
[bright92->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 |      
\end{verbatim}

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:

\textbf{Example}

\begin{verbatim}
[bright92->device[bright92]]% 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 |      
\end{verbatim}

The roughly 0B/s increase over 2 hours in the preceding output is a good sign.

Displaying union and intersection sets:  The \texttt{--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 \texttt{<lists>}, such as \texttt{-c, -r} and so on).

For example:
if the overlay \texttt{galeranodes} has the node \texttt{mon001} and the overlay \texttt{openstackhypervisors} has the nodes \texttt{node001, node002} then an example of a union of the set of these two overlays is:

\begin{verbatim}
[bright92->device]  
\end{verbatim}
Example

```bash
[bright92->device]% dumpmonitoringdata --union -3m now pageout -e galeranodes,openstackhypervisors
```

<table>
<thead>
<tr>
<th>Entity</th>
<th>Timestamp</th>
<th>Value</th>
<th>Info</th>
</tr>
</thead>
<tbody>
<tr>
<td>mon001</td>
<td>2018/09/11 11:31:56.198</td>
<td>192 KiB/s</td>
<td></td>
</tr>
<tr>
<td>mon001</td>
<td>2018/09/11 11:33:56.198</td>
<td>17.8 KiB/s</td>
<td></td>
</tr>
<tr>
<td>node002</td>
<td>2018/09/11 11:32:04.509</td>
<td>1.54 KiB/s</td>
<td></td>
</tr>
<tr>
<td>node002</td>
<td>2018/09/11 11:34:04.509</td>
<td>1.30 KiB/s</td>
<td></td>
</tr>
</tbody>
</table>

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

```bash
[bright92->device]% dumpmonitoringdata --union -3m now pageout -u -e galeranodes -n node001
```

<table>
<thead>
<tr>
<th>Entity</th>
<th>Timestamp</th>
<th>Value</th>
<th>Info</th>
</tr>
</thead>
<tbody>
<tr>
<td>mon001</td>
<td>1536659036.198</td>
<td>17.3 KiB/s</td>
<td></td>
</tr>
<tr>
<td>mon001</td>
<td>1536659156.198</td>
<td>116 KiB/s</td>
<td></td>
</tr>
<tr>
<td>node001</td>
<td>1536659008.997</td>
<td>1023.99 B/s</td>
<td></td>
</tr>
<tr>
<td>node001</td>
<td>1536659128.996</td>
<td>1.26 KiB/s</td>
<td></td>
</tr>
</tbody>
</table>

For an intersection of sets, the only syntax allowed is one that uses different grouping options:

Example

```bash
[bright92->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

```bash
[bright92->device[node001]]% dumpmonitoringdata -10m now threadsused:cmd
```

<table>
<thead>
<tr>
<th>Timestamp</th>
<th>Value</th>
<th>Info</th>
</tr>
</thead>
<tbody>
<tr>
<td>2017/07/21 11:16:00</td>
<td>42</td>
<td></td>
</tr>
<tr>
<td>2017/07/21 11:20:00</td>
<td>42</td>
<td></td>
</tr>
<tr>
<td>2017/07/21 11:22:00</td>
<td>41</td>
<td></td>
</tr>
<tr>
<td>2017/07/21 11:24:00</td>
<td>42</td>
<td></td>
</tr>
<tr>
<td>2017/07/21 11:26:00</td>
<td>42</td>
<td></td>
</tr>
</tbody>
</table>
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

[bright92->softwareimage] % dumpmonitoringdata -10m now nodesup default-image

<table>
<thead>
<tr>
<th>Timestamp</th>
<th>Value</th>
<th>Info</th>
</tr>
</thead>
<tbody>
<tr>
<td>2017/07/13 16:43:00</td>
<td>2</td>
<td></td>
</tr>
<tr>
<td>2017/07/20 17:37:00</td>
<td>2</td>
<td></td>
</tr>
</tbody>
</table>

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.

13.6.5 Monitoring Data Health Overview—The healthoverview Command

In figure 13.20, section 13.4.6, the Bright 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

[bright92->device]% healthoverview -n node00[1-3]

<table>
<thead>
<tr>
<th>Device</th>
<th>Sum</th>
<th>Maximum</th>
<th>Count</th>
<th>Age</th>
<th>Info</th>
</tr>
</thead>
<tbody>
<tr>
<td>node001</td>
<td>30</td>
<td>15</td>
<td>2</td>
<td>50.7s</td>
<td>hot, fan high</td>
</tr>
<tr>
<td>node002</td>
<td>30</td>
<td>15</td>
<td>2</td>
<td>50.7s</td>
<td>hot, fan high</td>
</tr>
<tr>
<td>node003</td>
<td>15</td>
<td>15</td>
<td>1</td>
<td>50.7s</td>
<td>hot</td>
</tr>
</tbody>
</table>

13.6.6 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
Monitoring: Monitoring Cluster Devices

```
[device]# monitoringinfo -n node001
```

<table>
<thead>
<tr>
<th>Service</th>
<th>Queued</th>
<th>Handled</th>
<th>Cache miss</th>
<th>Stopped</th>
<th>Suspended</th>
<th>Last operation</th>
</tr>
</thead>
<tbody>
<tr>
<td>Mon::CacheGather</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>yes</td>
<td>no</td>
<td></td>
</tr>
<tr>
<td>Mon::DataProcessor</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>yes</td>
<td>no</td>
<td></td>
</tr>
<tr>
<td>Mon::DataTranslator</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>no</td>
<td>no</td>
<td>Mon Jul 24 11:34:00</td>
</tr>
<tr>
<td>Mon::EntityMeasurableCache</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>no</td>
<td>no</td>
<td>Thu Jul 13 16:39:52</td>
</tr>
<tr>
<td>Mon::MeasurableBroker</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>no</td>
<td>no</td>
<td></td>
</tr>
<tr>
<td>Mon::Replicate::Collector</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>yes</td>
<td>yes</td>
<td></td>
</tr>
<tr>
<td>Mon::Replicate::Combiner</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>yes</td>
<td>yes</td>
<td></td>
</tr>
<tr>
<td>Mon::RepositoryAllocator</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>no</td>
<td>no</td>
<td></td>
</tr>
<tr>
<td>Mon::RepositoryTrim</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>yes</td>
<td>no</td>
<td></td>
</tr>
<tr>
<td>Mon::TaskInitializer</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>no</td>
<td>no</td>
<td>Thu Jul 13 16:39:52</td>
</tr>
<tr>
<td>Mon::TaskSampler</td>
<td>30</td>
<td>233,039</td>
<td>0</td>
<td>no</td>
<td>no</td>
<td>Mon Jul 24 11:34:00</td>
</tr>
<tr>
<td>Mon::Trigger::Actuator</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>yes</td>
<td>no</td>
<td></td>
</tr>
<tr>
<td>Mon::Trigger::Dispatcher</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>yes</td>
<td>no</td>
<td></td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Cache</th>
<th>Size</th>
<th>Updates</th>
<th>Requests</th>
</tr>
</thead>
<tbody>
<tr>
<td>ConsolidatorCache</td>
<td>0</td>
<td>17</td>
<td>0</td>
</tr>
<tr>
<td>EntityCache</td>
<td>10</td>
<td>17</td>
<td>935,280</td>
</tr>
<tr>
<td>GlobalLastRawDataCache</td>
<td>87</td>
<td>17</td>
<td>0</td>
</tr>
<tr>
<td>LastRawDataCache</td>
<td>142</td>
<td>17</td>
<td>427,301</td>
</tr>
<tr>
<td>MeasurableCache</td>
<td>231</td>
<td>17</td>
<td>935,230</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Cache</th>
<th>Up</th>
<th>Down</th>
<th>Closed</th>
</tr>
</thead>
<tbody>
<tr>
<td>DeviceStateCache</td>
<td>3</td>
<td>0</td>
<td>0</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Replicator</th>
<th>First</th>
<th>Last</th>
<th>Requests</th>
<th>Samples</th>
<th>Sources</th>
</tr>
</thead>
<tbody>
<tr>
<td>ReplicateRequestHandler</td>
<td>-</td>
<td>-</td>
<td>0</td>
<td>0</td>
<td></td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Cache</th>
<th>Queued</th>
<th>Delivered</th>
<th>Handled</th>
<th>Pickup</th>
</tr>
</thead>
<tbody>
<tr>
<td>Cache</td>
<td>0</td>
<td>120</td>
<td>932,257</td>
<td>7,766</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Plotter</th>
<th>First</th>
<th>Last</th>
<th>Count</th>
<th>Samples</th>
<th>Sources</th>
<th>Requests</th>
</tr>
</thead>
<tbody>
<tr>
<td>RequestDispatcher</td>
<td>-</td>
<td>-</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>-</td>
</tr>
<tr>
<td>RequestHandler</td>
<td>-</td>
<td>-</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>-</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Storage</th>
<th>Elements</th>
<th>Disk size</th>
<th>Usage</th>
<th>Free disk</th>
</tr>
</thead>
<tbody>
<tr>
<td>Mon::Storage::Engine</td>
<td>0</td>
<td>0 B</td>
<td>0.0%</td>
<td>-</td>
</tr>
<tr>
<td>Mon::Storage::Message</td>
<td>0</td>
<td>0 B</td>
<td>0.0%</td>
<td>-</td>
</tr>
<tr>
<td>Mon::Storage::RepositoryId</td>
<td>0</td>
<td>0 B</td>
<td>0.0%</td>
<td>-</td>
</tr>
</tbody>
</table>

13.6.7 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 Bright 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
13.6 Obtaining Monitoring Data Values

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@bright92 ~]# cmsh
[bright92]% 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@bright92 ~]# cmsh
[bright92]% device use node001
[...->device[node001]]% latestmonitoringdata | grep ssh2node
ssh2node Network PASS 43m 38s filtered
[...->device[node001]]% monitoringdrop --dry-run --filtered
Entity      Measurable
------------ ------------
node001     ssh2node
```

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@bright92 ~]# cmsh
[bright92]% 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.
13.6.8 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

<table>
<thead>
<tr>
<th>Service</th>
<th>Queued</th>
<th>Handled</th>
<th>Cache miss</th>
<th>Stopped</th>
<th>Suspended</th>
</tr>
</thead>
<tbody>
<tr>
<td>Mon::CacheGather</td>
<td>425</td>
<td>39,857</td>
<td>0</td>
<td>no</td>
<td>yes</td>
</tr>
<tr>
<td>Mon::DataConverter</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>no</td>
<td>yes</td>
</tr>
<tr>
<td>Mon::DataProcessor</td>
<td>0</td>
<td>6,609,369</td>
<td>0</td>
<td>no</td>
<td>yes</td>
</tr>
<tr>
<td>Mon::DataTranslator</td>
<td>0</td>
<td>311,658</td>
<td>0</td>
<td>no</td>
<td>yes</td>
</tr>
<tr>
<td>Mon::EntityMeasurableCache</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>no</td>
<td>yes</td>
</tr>
<tr>
<td>Mon::MeasurableBroker</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>no</td>
<td>yes</td>
</tr>
<tr>
<td>Mon::PerpetualTaskManager</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>no</td>
<td>yes</td>
</tr>
<tr>
<td>Mon::Replicate::Collector</td>
<td>0</td>
<td>0</td>
<td>yes</td>
<td>yes</td>
<td>yes</td>
</tr>
<tr>
<td>Mon::Replicate::Combiner</td>
<td>0</td>
<td>0</td>
<td>yes</td>
<td>yes</td>
<td>yes</td>
</tr>
</tbody>
</table>
...

[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
04:42:00 PM 4.04 0.00 4.04 0.00 96.97
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 `monitorsuspend`, 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 865).

**Disabling job information collection completely**: by setting the value of the `JobInformationDisabled` directive to 0 (page 870).

**For the Slurm workload manager only, disabling job accounting**: by setting the value of the `SlurmDisableAccountingParsing` directive to 0 (page 865).

**Reducing the duration for which job data is stored**: by reducing the value of the `JobInformationKeepDuration` (page 870).

### 13.6.9 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

```
[bright92] % device
[bright92->device] % monitoringpickup
Hostname Interval Times Priority
----------------- ----------- -------- --------
bright92 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

```
[bright92] % device use node001
[bright92->device[node001]] % monitoringpickup --interval 1m
Changed 1 pickup intervals
```

```
[bright92->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

```bash
[bright92] device use node001
[bright92->device[node001]] monitoringpickup --interval 1m --times 10
```

Changed 1 pickup intervals

```
[bright92->device[node001]] monitoringpickup
```

<table>
<thead>
<tr>
<th>Hostname</th>
<th>Interval</th>
<th>Times</th>
<th>Priority</th>
</tr>
</thead>
<tbody>
<tr>
<td>node001</td>
<td>1m</td>
<td>10</td>
<td>100</td>
</tr>
</tbody>
</table>

The --forever option lets the pickup be carried out “forever”.

Example

```
[bright92->device[node001]] monitoringpickup --interval 30s --forever
```

Changed 1 pickup intervals

```
[bright92->device[node001]] monitoringpickup
```

<table>
<thead>
<tr>
<th>Hostname</th>
<th>Interval</th>
<th>Times</th>
<th>Priority</th>
</tr>
</thead>
<tbody>
<tr>
<td>node001</td>
<td>30s</td>
<td>-</td>
<td>100</td>
</tr>
</tbody>
</table>

The --priority option applies the priority to equal or lower priority settings:

Example

```
[bright92] device
[bright92->device] monitoringpickup -n node00[1-2]
```

```
[bright92->device] monitoringpickup -n node00[1-2] --interval 5s --priority 50
```

Changed 1 pickup intervals

```
[bright92->device] monitoringpickup -n node00[1-2]
```

<table>
<thead>
<tr>
<th>Hostname</th>
<th>Interval</th>
<th>Times</th>
<th>Priority</th>
</tr>
</thead>
<tbody>
<tr>
<td>node001</td>
<td>1m</td>
<td>12</td>
<td>80</td>
</tr>
<tr>
<td>node002</td>
<td>5s</td>
<td>1</td>
<td>50</td>
</tr>
</tbody>
</table>

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:

```
[bright92->device] monitoringpickup -n node00[1-2]
```

```
[bright92->device] monitoringpickup -n node00[1-2] --interval 2m
```

<table>
<thead>
<tr>
<th>Hostname</th>
<th>Interval</th>
<th>Times</th>
<th>Priority</th>
</tr>
</thead>
<tbody>
<tr>
<td>node001</td>
<td>1m</td>
<td>12</td>
<td>80</td>
</tr>
<tr>
<td>node002</td>
<td>2m</td>
<td>0</td>
<td>0</td>
</tr>
</tbody>
</table>

\(^3\)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.

[bright@...->jobmetricsettings] % show
Parameter Value
------------------ ------
...  
Pickup interval 5s
Pickup priority 50
Pickup times 12

13.7 Offloaded Monitoring

Offloaded monitoring is a feature introduced in NVIDIA Bright 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 13.7.3.

13.7.1 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. How-
ever, 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.
13.7.2 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

```
[bright92->device]% use node001
[bright92->device[node001]% roles
[bright92->device[node001]->roles]% assign monitoring
[bright92->device*[node001*]->roles*[monitoring*]]% show
```

```
<table>
<thead>
<tr>
<th>Parameter</th>
<th>Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>Name</td>
<td>monitoring</td>
</tr>
<tr>
<td>Revision</td>
<td></td>
</tr>
<tr>
<td>Type</td>
<td>MonitoringRole</td>
</tr>
<tr>
<td>Add services</td>
<td>yes</td>
</tr>
<tr>
<td>Provisioning associations</td>
<td>&lt;0 internally used&gt;</td>
</tr>
<tr>
<td>Number of backups</td>
<td>2</td>
</tr>
<tr>
<td>Backup ring</td>
<td>automatic</td>
</tr>
</tbody>
</table>
```

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.

13.7.3 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 18. By default, a monitoring role is then assigned to both the head nodes.

This has the effect of doubling the monitoring capacity of the head node pair in NVIDIA Bright Cluster Manager 9.1, in comparison with a head node pair in NVIDIA Bright 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 229), 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):

![Schematic of monitoring sampling flow](image)

Figure 13.24: 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 13.7.2) is used to configure the number of backups. In the following schematic, two neighboring monitoring nodes are used as backup:

![Schematic of backup flow](image)

Figure 13.25: 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 Bright 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:

![Diagram showing backup nodes](image)

Figure 13.26: A More Sophisticated Backup Flow For Offloaded Monitoring With Dedicated Monitoring Nodes And Dedicated Backup Nodes

Backup nodes for the monitoring nodes take away the monitoring data backup task from the monitoring nodes. This frees up the monitoring nodes so that they can take on even more monitoring.

**Provisioning role on monitoring nodes:** If there is enough capacity on the dedicated monitoring nodes, and the cluster spends most of its time in a relatively steady state where its nodes do not reboot frequently, then adding a provisioning role to the monitoring nodes can be an efficient use of resources. In this case the monitoring nodes are obviously not so dedicated, but the advantage is that rebooting the entire cluster is then faster, at the cost of perhaps some extra load on the monitoring nodes during such a reboot.

**Offloaded Monitoring Sampling And Backup Flows For Edge Computing**

For a cluster with edge configured, the edge director flows in the edge network are analogous to head node flows in the local network. Thus, monitoring is carried out by the directors on the edge nodes, and the directors also sample themselves.

Thus, edge directors, not in a high-availability configuration, have the monitoring sampling data flow shown by the following schematic (figure 13.27):
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 13.28):

The backup data flow for a non-HA configuration would then be as follows for an edge director (figure 13.29):

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 13.30):

![Diagram](image_url)

Figure 13.30: 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

### 13.8 The User Portal

#### 13.8.1 Accessing The User Portal

The user portal is compatible with most browsers using reasonable settings. For Internet Explorer, version 9 or later is required.

The user portal is located by default on the head node. For a head node `bright92`, it is accessible to users for a login via a browser at the URL `http://bright92`, or more directly via `https://bright92:8081/userportal`. The state of the cluster can then be viewed by the users via an interactive interface.

The first time a browser is used to login 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 Bright Computing, and the attributes of the cluster owner are part of the certificate. However, Bright Computing 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 Bright Computing 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.
13.8.2 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@bright92 ~]# 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 Bright View

Example

```
[root@bright92 ~]# cmsh
[bright92]# user
[bright92->user%] add forrestgump
[bright92->user[forrestgump]%] set password
enter new password:
retype new password:
[bright92->user*[forrestgump*]%] commit
[bright92->user*[forrestgump*]%] quit
```

• the common username and password should be set in the appropriate place in the configuration file, common-credentials.json:

Example

```
[root@bright92 ~]# 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 Bright 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.

13.8.3 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.
13.8.4 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 13.31):

![User Portal: Overview Page](image)

Figure 13.31: 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. Monitoring and job accounting charts can be viewed on clicking upon the associated icons, ![Chart Icon] and ![Graph Icon], at the top right corner of the user portal page.

User Portal Monitoring Page

The user portal's Monitoring page allows charts of measurables to be plotted in a monitoring window in a very similar manner to how it is done in section 13.3.1.

User Portal Job Accounting Page

The user portal's Accounting and reporting page allows job accounting to be viewed in an accounting panel in a very similar manner to how it is done in section 15.5.
13.9 Cloud Job Tagging

Cloud job tagging is about the ability for cloud job instances to have their associated cloud resources tagged. This is only possible for AWS at the time of writing (February 2020). Enabling cloud job tagging via NVIDIA Bright 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 (section 15.2). AWS cloud resource tagging is only active and handled within AWS.

Cloud job tags allow the time span between tag creation and removal to be associated with a particular workload on the node.

In cmsh, for a cloud node, cloud job tagging can be enabled within cloud mode by setting the cloudjobtagging parameter for the EC2Provider entity to yes.

Example

```
cmsh -c 'cloud; use amazon; set cloudjobtagging yes; commit'
```

If it is set to yes, then every job running on a cloud node using that specific provider is tagged according to the applied tags.

A subset of the tags for cloud jobs are cost allocation tags. Cloud job cost allocation tags allow AWS costs to be tracked for jobs. A cost allocation tag can be:

- an AWS generated tag: defined, created, and applied by AWS
- a user-defined tag: defined, created, and applied by the user

By default, 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:

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

13.10 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.
13.10.1 Viewing Events In Bright View

In Bright View, events can be viewed by clicking on the Events icon of figure 13.5. 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.

13.10.2 Viewing Events In cmsh

The events command is a global cmsh command. It allows events to be viewed at several severity levels (section 13.2.7), 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

```
[bright92->device]% events
Private events: off
Broadcast events: on
Level: notice
```

Running the command with options allows the viewing and setting of events as follows:

- **on [broadcast|private]**: event messages are displayed as they happen in a session, with cmsh prompts showing in between messages:
  - If only `on` is set, then all event messages are displayed as they happen:
    - either to all open cmsh sessions, and also in Bright 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 Bright 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**

```
[bright92->softwareimage]% events on private
Private events: on
[bright92->softwareimage]% updateprovisioners
Provisioning nodes will be updated in the background.
[bright92->softwareimage]%
Tue Apr 29 01:19:12 2014 [notice] bright92: Provisioning started: sendi...
[bright92->softwareimage]%
Tue Apr 29 01:19:52 2014 [notice] bright92: Provisioning completed: sen...
updateprovisioners [ COMPLETED ]
[bright92->softwareimage]%
#events were indeed seen in cmsh session
[bright92->softwareimage]%
#now block the events and rerun update:
[bright92->softwareimage]%
#events off private
Private events: off
[bright92->softwareimage]% updateprovisioners
Provisioning nodes will be updated in the background.
[bright92->softwareimage]%
#let this 2nd update run for a while
[bright92->softwareimage]%
#(time passes)
[bright92->softwareimage]%
#nothing seen in cmsh session.
[bright92->softwareimage]%
#show a 2nd update did happen:
[bright92->softwareimage]%
events 4 | grep -i provisioning
Tue Apr 29 01:19:12 2014 [notice] bright92: Provisioning started: sendi...
Tue Apr 29 01:19:52 2014 [notice] bright92: Provisioning completed: sen...
```
13.10.3 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@bright92 ~]# 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”.

13.10.4 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 13.2.7), 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 Bright View event viewer as shown in figure 13.32:

![Event Bucket Message Example](image)

Figure 13.32: Bright View Monitoring: Event Bucket Message Example

Event Bucket Filter

Regexp 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, bright92, 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 bright92"
bright92 .................. (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 Bright 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
   ```
13.11 Monitoring Location With GNSS

GNSS (Global Navigation Satellite System) is the term given to GPS and similar systems. GNSS can be used to allow devices with the appropriate GNSS hardware to work out their location. The hardware is commonly implemented as a PCI-X card. A use case for this is to allow an engineer to walk to a node with a mobile phone, or to determine a sensible provisioning host.

The hardware requires the ability to receive satellite signals via an antenna. From the signals, the time of receipt and location can be worked out. The cluster manager makes the results available in the locations submode of the base partition of cmsh:

Example

```
[bright92->partition[base]]% locations
```

<table>
<thead>
<tr>
<th>Type</th>
<th>Entity</th>
<th>Age</th>
<th>Latitude</th>
<th>Longitude</th>
<th>Height</th>
<th>Message</th>
</tr>
</thead>
<tbody>
<tr>
<td>EdgeSite</td>
<td>Amsterdam West</td>
<td>2d 17h</td>
<td>52.1904</td>
<td>4.939</td>
<td>0</td>
<td>Amsterdam-South</td>
</tr>
<tr>
<td>EdgeSite</td>
<td>Fort-Collins</td>
<td>6d 7h</td>
<td>40.5538</td>
<td>-105.0849</td>
<td>0</td>
<td>Fort-Collins</td>
</tr>
<tr>
<td>HeadNode</td>
<td>rima</td>
<td>57m 3s</td>
<td>52.3927</td>
<td>4.8361</td>
<td>0</td>
<td>Amsterdam</td>
</tr>
<tr>
<td>PhysicalNode</td>
<td>bright-office-director</td>
<td>2d 16h</td>
<td>52.1903</td>
<td>4.9163</td>
<td>0</td>
<td>Amsterdam-South</td>
</tr>
<tr>
<td>PhysicalNode</td>
<td>bright-office-node001</td>
<td>2d 17h</td>
<td>52.1904</td>
<td>4.9617</td>
<td>0</td>
<td>Amsterdam-South</td>
</tr>
<tr>
<td>PhysicalNode</td>
<td>fort-collins-director</td>
<td>6d 7h</td>
<td>40.5538</td>
<td>-105.0849</td>
<td>0</td>
<td>Fort-Collins</td>
</tr>
</tbody>
</table>

In practice, due to environmental interference, a minimum resolution of 5m is common for longitude and latitude. 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.

13.12 Monitoring Report Queries

13.12.1 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

```
[bright92->monitoring->report]% fields | head -30
```

<table>
<thead>
<tr>
<th>Name</th>
<th>Type</th>
<th>Values</th>
</tr>
</thead>
<tbody>
<tr>
<td>AlertLevel</td>
<td>METRIC</td>
<td></td>
</tr>
<tr>
<td>BIOS Date</td>
<td>SYSINFO</td>
<td>04/01/2014</td>
</tr>
<tr>
<td>BIOS Vendor</td>
<td>SYSINFO</td>
<td>SeaBIOS</td>
</tr>
<tr>
<td>BIOS Version</td>
<td>SYSINFO</td>
<td>SeaBIOS</td>
</tr>
<tr>
<td>BlockedProcesses</td>
<td>METRIC</td>
<td></td>
</tr>
<tr>
<td>BufferMemory</td>
<td>METRIC</td>
<td></td>
</tr>
<tr>
<td>BytesRecv</td>
<td>COUNTER</td>
<td></td>
</tr>
<tr>
<td>BytesSent</td>
<td>COUNTER</td>
<td></td>
</tr>
<tr>
<td>...</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
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**

```
[bright92->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**

```
[bright92->monitoring->report]% execute -q dualcpu
cores processor vendor Hostnames
-------- ---------------- ----------------------------------------------
2 GenuineIntel nas,bright92-a,bright92-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**

```
[bright92->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
[bright92->monitoring->report]%
```

Alternatively, the query can be saved and run as follows:

**Example**

```
[bright92->monitoring->report]% add ssh2node-category
[bright92->monitoring->report*][ssh2node-category*]% set query

the editor opens up and the line
execute filter status == up group_by ssh2node category

is entered

[bright92->monitoring->report*[ssh2node-category*]% commit
[bright92->monitoring->report*[ssh2node-category*]% execute
ssh2node category Hostnames
-------- -------- -----------------------
PASS default node001,node002
FAIL default node003,node004
PASS gpu gpu01..gpu20
[bright92->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

[bright92->monitoring->report]% execute filter status == up group_by ssh2node category --save /tmp/test
[bright92->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

[bright92->monitoring->report]% device power status -n `/tmp/PASS-default.lst`
custom ................. [ ON     ] node001
custom ................. [ ON     ] node002
14 Monitoring: Job Monitoring

14.1 Job Metrics Introduction

Most HPC administrators set up device-centric monitoring to keep track of cluster node resource use. This means that metrics are selected for devices, and the results can then be seen over a period of time. The results can be viewed as a graph or data table, according to the viewing option chosen. This is covered in Chapter 13.

The administrator can also select a job that is currently running, or that has recently run, and get metrics for nodes, memory, CPU, storage, and other resource use for the job. This is known as job monitoring, which is, as the term suggests, about job-centric rather than device-centric monitoring. Job monitoring is covered in this chapter, and uses job metrics.

For perspective, monitoring as discussed until now has been based on using devices or jobs as the buckets for which resource use values are gathered. Administrators can also gather, for resources consumed by jobs, the resources used by users (or any other aggregation entity) as the buckets for the values. This is typically useful for watching over the resources used by a user (or other aggregation entity) when jobs are run on the cluster. User-centric monitoring—or more generally, aggregation-centric monitoring—for jobs is termed job accounting and is covered in Chapter 15.

14.2 Job Metrics With Cgroups

Job metrics collection uses control groups (cgroups), (section 7.10). Each job is associated with a specific cgroup that is created in each of the three base cgroups that are associated with particular cgroup controllers. The cgroup controllers are kernel components that allow metrics to be collected for processes. The PIDs of these processes are in the cgroups tasks file.

NVIDIA Bright Cluster Manager 9.2 uses the following cgroup controllers:

- `blkio`: provides block device metrics,
- `cpuacct`: provides CPU usage metrics,
- `memory`: provides memory usage metrics.

In NVIDIA Bright Cluster Manager before version 9.1, each job had to be put by a workload manager into a unique cgroup. However, with NVIDIA Bright Cluster Manager 9.1 onwards, this no longer necessary. By default, the cluster manager still configures all supported workload managers to run jobs in cgroups, but it is now CMDaemon that manages the cgroup life cycle. Thus, CMDaemon ensures that:

- the necessary cgroups are created per job
- ensures that the cgroups are removed after the job is finished
- and that the last values of the metrics are collected.
Even if the administrator completely disables cgroups management in the workload manager, CMDaemon can still create and remove the three cgroups associated with the job, with each of those cgroups associated with one of the three previously-mentioned cgroup controllers.

If the workload manager creates some (or all three) cgroups for a job, then CMDaemon does not try to recreate the cgroup, but does take charge of the removal of cgroups.

In NVIDIA Bright Cluster Manager before version 9.1, `cm-wlm-setup` configured systemd to use a joined cgroup with the following parameter settings:

**Example**

```
[root@node001 ~]# grep JoinControllers /etc/systemd/system.conf
JoinControllers=blkio,cpuacct,memory,freezer
[root@node001 ~]#
```

Currently this is not needed. However, if this setting remains, then CMDaemon can still collect job metrics. In order to reset the cgroup layout to the default one, the administrator can run:

```
cm-wlm-setup --reset-cgroups
```

This command removes the `JoinControllers` parameter and regenerates initrd. A reboot of the nodes is required after this.

When a job is started CMDaemon detects all the job processes. CMDaemon then ensures that the required cgroups are created, and allocates the detected processes to those cgroups. CMDaemon does not configure the cgroups in any way—this is the responsibility of the workload manager.

The tables in Appendix G.1.6 list the job metrics that the cluster manager can monitor and visualize. If job metrics are set up (section 14.4), then:

1. on virtual machines, block device metrics may be unavailable because of virtualization.
2. for now, the metrics are retrieved from cgroups created by the workload manager for each job. When the job is finished the cgroup is removed from the filesystem along with all the collected data. Retrieving the jobs metric data therefore means that CMDaemon must sample the cgroup metrics before the job is finished. If CMDaemon is not running during a time period for any reason, then the metrics for that time period cannot be collected, even if CMDaemon starts later.
3. block device metrics are collected for each block device by default. Thus, if there are \( N \) block devices, then there are \( N \) collected block device metrics. The monitored block devices can be excluded by configuration as indicated in section 14.4.

### 14.3 Job Information Retention

Each job adds a set of metric values to the monitoring data. The longer a job runs, the more data is added to the data. By default, old values are cleaned up from the database in order to limit its size. In NVIDIA Bright Cluster Manager 9.2 there are several advanced configuration directives to control the job data retention, with names and default values as follows:

<table>
<thead>
<tr>
<th>Advanced Configuration Directive</th>
<th>Default value</th>
<th>Unit</th>
</tr>
</thead>
<tbody>
<tr>
<td>JobInformationDisabled</td>
<td>0</td>
<td></td>
</tr>
<tr>
<td>JobInformationKeepDuration</td>
<td>2419200</td>
<td>s</td>
</tr>
<tr>
<td>JobInformationKeepCount</td>
<td>8192</td>
<td></td>
</tr>
<tr>
<td>JobInformationMinimalJobDuration</td>
<td>0</td>
<td>s</td>
</tr>
<tr>
<td>JobInformationFlushInterval</td>
<td>600</td>
<td>s</td>
</tr>
</tbody>
</table>

These directives are described in detail in Appendix C, page 870.
14.4 Job Metrics Sampling Configuration

Job metrics sampling can be configured to varying degrees. For clusters where hundreds of thousands of jobs are run in a day it often makes little sense to monitor jobs, and it is often helpful to disable the JobSampler and JobMetadataSampler data producers:

Example

```
[drg@bright92->monitoring->setup]% set jobsampler disabled yes
[drg@bright92->monitoring->setup]% set jobmetadatasampler disabled yes
[drg@bright92->monitoring->setup]% commit
```

An alternative is to use the equivalent CMDaemon directive JobInformationDisabled, as explained on page 643.

If however CMDaemon is to keep the monitoring data, then the collection of job metrics is carried out from the cgroups in which a job runs. The administrator can tune some low level metric collection options for the JobSampler data producer in the jobmetricsettings submode:

Example

```
[drg@bright92% monitoring setup]
[drg@bright92->monitoring->setup]% use jobsampler
[drg@bright92->monitoring->setup[JobSampler]% jobmetricsettings
[drg@bright92->monitoring->setup[JobSampler]->jobmetricsettings]% show
```

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>CGroup base directory</td>
<td>/sys/fs/cgroup</td>
</tr>
<tr>
<td>CGroup search</td>
<td>freezer</td>
</tr>
<tr>
<td>Enable Advanced Metrics</td>
<td>no</td>
</tr>
<tr>
<td>Exclude Devices</td>
<td>loop,sr</td>
</tr>
<tr>
<td>Exclude Metrics</td>
<td></td>
</tr>
<tr>
<td>Include Devices</td>
<td></td>
</tr>
<tr>
<td>Include Metrics</td>
<td></td>
</tr>
<tr>
<td>Keep alive sleep</td>
<td>8w</td>
</tr>
<tr>
<td>No GPUs is all GPUs</td>
<td>no</td>
</tr>
<tr>
<td>Pickup interval</td>
<td>5s</td>
</tr>
<tr>
<td>Pickup priority</td>
<td>50</td>
</tr>
<tr>
<td>Pickup times</td>
<td>12</td>
</tr>
<tr>
<td>Sampling Type</td>
<td>Both</td>
</tr>
</tbody>
</table>

The configuration parameters are:

<table>
<thead>
<tr>
<th>Parameter Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>CGroup base directory</td>
<td>Cgroup base directory (default: /sys/fs/cgroup)</td>
</tr>
<tr>
<td>CGroup search</td>
<td>Search field for finding the workload manager cgroup (default: freezer)</td>
</tr>
</tbody>
</table>

...continues
### Monitoring: Job Monitoring

...continued

<table>
<thead>
<tr>
<th>Parameter Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Enable Advanced Metrics</td>
<td>Enables advanced metrics</td>
</tr>
<tr>
<td>Exclude Devices</td>
<td>Block devices for which job metrics will not collect metrics</td>
</tr>
<tr>
<td>Include Devices</td>
<td>If the list is not empty then only block device metrics for these devices will be collected, while for other devices the metrics will be skipped</td>
</tr>
<tr>
<td>Enable Advanced</td>
<td>Indicates whether advanced job metrics should be enabled (options: true, false)</td>
</tr>
<tr>
<td>Exclude Metrics</td>
<td>List of metric names that should not be collected</td>
</tr>
<tr>
<td>Include Metrics</td>
<td>List of metric names that should be added to metric collection</td>
</tr>
<tr>
<td>Keep alive sleep</td>
<td>Time the cgroup keepalive process sleeps (default: 8 weeks)</td>
</tr>
<tr>
<td>No GPUs is all GPUs</td>
<td>Include data from all GPUs even if the job has no GPUs defined (default: no)</td>
</tr>
<tr>
<td>Pickup interval</td>
<td>Initially higher pickup interval (default: 5s). By default this settles down to the normal pickup interval (with a default of 120s) after the value of Pickup times has been exceeded.</td>
</tr>
<tr>
<td>Pickup priority</td>
<td>Priority of the pickup interval change (default: 50)</td>
</tr>
<tr>
<td>Pickup times</td>
<td>Number of times to apply the initially higher pickup interval (default: 12)</td>
</tr>
<tr>
<td>Sampling Type</td>
<td>Type of metric sampling (default: Both)</td>
</tr>
</tbody>
</table>

The amount of monitoring data gathered can also be reduced by reducing the Maximal age and Maximal samples for data producers (section 13.4.1) to smaller, but still non-zero values. A way to do this is described in section 17.8.4.

### 14.4.1 The Job Metrics Collection Processing Mechanism

The cm-cgroup-job-keepalive Process

From NVIDIA Bright Cluster Manager 9.0 onwards, when a WLM job starts, CMDaemon tracks the moment it starts and finishes, and is able to collect metrics for it. As part of this enhanced jobs metrics collection, a keepalive process, cm-cgroup-job-keepalive, is run for each job. Each keepalive process is a temporary process, and is added by CMDaemon to the same cgroup that the original job was placed in by the WLM.
The process `cm-cgroup-job-keepalive` itself does no work. It sleeps, and by existing it prevents its cgroup being deleted.

After a job is finished, the workload manager would normally remove the related cgroup. However, the existence of the `cm-cgroup-job-keepalive` process prevents the deletion. This allows CMDaemon to collect the very last metrics data for the job from the cgroup when the job finishes. The CMDaemon then stops the `cm-cgroup-job-keepalive` process, and the cgroup is then removed because it is no longer needed.

When CMDaemon starts the `cm-cgroup-job-keepalive` process for a job, it passes the appropriate job ID, and how long it can run, in its command line options. Those values are not used by the `cm-cgroup-job-keepalive` process itself, but they are convenient for seeing what job the process is running for, and how long the job has run since it was started. For example, for a Slurm job with id 2 the running cgroup keeper process could look like:

Example

```
[root@node001 ~]# ps auxf | tail -n 10 | cut -b18-45 --complement
root  1954  0 S   18:12 0:03 _ /cm/local/apps/cmd/sbin/cmd -s -n -P /var/run/cmd.pid
root  2632  0 Ss  18:15 0:00 _ /cm/local/apps/cmd/sbin/cm-cgroup-job-keepalive --job 2 8w
root  2241  0 S   18:13 0:00 /cm/shared/apps/slurm/18.08.4/sbin/slurmd
root  2627  0 Sl  18:15 0:00 slurmstepd: [2.batch]
cmsuppo+ 2631  0 S   18:15 0:00 _ /bin/bash /cm/local/apps/slurm/var/spool/job00002/slurm_script
cmsuppo+ 2642  0 S   18:15 0:00 _ /cm/shared/apps/stresscpu/current/stresscpu2
cmsuppo+ 2643  0 S   18:15 0:00 _ /cm/shared/apps/stresscpu/current/stresscpu2
cmsuppo+ 2644 98 R  18:15 4:46 _ /cm/shared/apps/stresscpu/current/stresscpu2
[root@node001 ~]#
```

(The `cut` command is just used in the example to cut out the middle bits of the output so that it fits the page format well).

The Keep Alive Sleep Time

By default the cgroup keeper process stops after 8 weeks. This value should be increased if the jobs that are expected to run will take longer than 8 weeks. The value can be set in the `Keep Alive Sleep` parameter of the job metrics settings. If a job runs for longer than the value of `Keep Alive Sleep`, then CMDaemon cannot collect the very last metrics (from around the time that the job has finished). However all other metrics will be collected for the job as expected, even if the job running time exceeds the `Keep Alive Sleep` time.

The OOB intervals Parameter

When metric collection for a new job has just started, CMDaemon samples more frequently than later on. This more frequent sampling behaviour is defined by the parameter `OOB intervals` (out of band sampling interval) in the data producer configuration. In the case of job metrics collection this more frequent sampling behaviour is in `JobSampler`.

By default, the sampling interval returns to the standard `Interval` value (with a default value of 120s), as defined in the data producer settings, after the value of `Pickup times` (with a default value of 12) has been exceeded.

The parameter `Exclude Metrics` can be used to exclude metrics that are currently enabled. For example, if advanced metrics collection is enabled then `Exclude Metrics` allows either default or advanced metrics to be excluded by name.

14.5 Job Monitoring In cmsh

The following commands are associated with monitoring job measurable within jobs submode (`cmsh→wlm<[workload manager]→jobs`, section 7.7):

```
The `measurables` Command

A list of job-associated measurables can be seen in the jobs submode (`cmsh→[<workload manager>]→jobs`) using the `measurables` command with a job ID. For example (much output elided):

**Example**

```
[bright92->wlm[slurm]→jobs]% measurables 26
blkio.io_service_bytes_total
... memory.usage
```

A list of node-associated measurables can also be seen if the `-n` option is used (much output elided):

**Example**

```
[bright92->wlm[slurm]→jobs]% measurables -n 26
...
gpu_power_usage:gpu0
... memory.usage
```

The `filter` Command

The `filter` command provides filtered historic job-related information. It does not provide measurables data. The command and its options can be used to:

- retrieve running, pending, failed or finished jobs information
- select job data using regular expressions to filter by job ID
- list jobs by user name, user group, or workload manager

The `--limit` option can be used to limit the number of results displayed

**Example**

```
[bright92->wlm[slurm]→jobs]% filter -n mgbench --ended --limit 2
Job ID  Job name  User  Queue  Submit time  Start time  End time  Nodes  Exit code
-------  --------  -----  ------  --------------  --------------  --------------  ------  --------
26      mgbench  alice  defq  May 14 11:21:41  May 14 11:36:36  May 14 11:50:24  node001  0
27      sleep    bob  defq  May 14 11:23:00  May 14 11:50:24  May 14 12:00:25  node001  0
```

The data shown is retrieved from the running workload manager, as well as from the accounting file or database maintained by the workload manager.

Further details on the options to `filter` can be seen by running `help filter`.

The `info` Command

A handy command to obtain job information is `info`, followed by the job number:

**Example**

```
[bright92->wlm[slurm]→jobs]% info 26
Parameter              Value
----------------------- -----------------------------------------------
Revision               WlmCluster slurm
Job ID                 26
Job name               mgbench
```
14.5 Job Monitoring In cmsh

Persistent no
Queue defq
User alice
Group alice
Account
Parent ID
GGroup slurm/uid_1002/job_26
Nodes node001
Submit time 14/05/2020 11:21:41
Start time 14/05/2020 11:36:36
End time 14/05/2020 11:50:24
Exit code 0
Status COMPLETED

The dumpmonitoringdata Command

The dumpmonitoringdata command displays data for measurables in the jobs submode (cmsh→<workload manager>→jobs). It is very similar to the dumpmonitoringdata command for measurables in device mode (section 13.6.4). The main difference in the behavior of dumpmonitoringdata for these modes is that:

• In the jobs submode it shows monitoring data over a period of time for a specified job ID

• In device mode (cmsh→device) it shows monitoring data over a period of time for a specified device.

A less obvious difference is that:

• In the jobs submode the start and end time for the monitoring data for the job does not need to be specified. By default the start and end time of the job is assumed.

• In device mode the start and end time for the monitoring data for the device must be specified.

The usage of the dumpmonitoringdata command for job measurables is:

dumpmonitoringdata [OPTIONS] [<start-time> <end-time>] <measurable> <job ID>

Options allow measurables to be retrieved and presented in various ways, including by maximum value, raw or interpolated data, and human-friendly forms. The user can also specify custom periods for the options.

For example, the historical job with job ID 26 might display output for the job-associated measurable memory.usage as follows (output elided):

Example

[bright92->wlm[slurm]->jobs]% dumpmonitoringdata memory.usage 26
Start: Thu May 14 11:36:36 2020
End: Thu May 14 11:50:24 2020
Nodes: node001

<table>
<thead>
<tr>
<th>Timestamp</th>
<th>Value</th>
<th>Info</th>
</tr>
</thead>
<tbody>
<tr>
<td>2020/05/14 11:36:37</td>
<td>708 KiB</td>
<td></td>
</tr>
<tr>
<td>2020/05/14 11:36:38</td>
<td>117 MiB</td>
<td></td>
</tr>
<tr>
<td>2020/05/14 11:36:39</td>
<td>29.8 MiB</td>
<td></td>
</tr>
<tr>
<td>...</td>
<td></td>
<td></td>
</tr>
<tr>
<td>2020/05/14 11:50:24</td>
<td>86.6 MiB</td>
<td></td>
</tr>
</tbody>
</table>

[bright92->wlm[slurm]->jobs]%
The start and end times are optional, so specifying them is typically unnecessary. If they are not specified, then the data values over the entire period of the job run are displayed.

That job with ID 26, for the example used in this section 14.5, happens to be mgbench, a GPU benchmarking program. So displaying output for the device-associated measurable, gpu_power_usage, during the job run can also be useful:

**Example**

```
[bright92->wlm[slurm]->jobs]% dumpmonitoringdata gpu_power_usage:gpu0 26
Start: Thu May 14 11:36:36 2020
End: Thu May 14 11:50:24 2020
Nodes: node001
Timestamp Value Info
-------------------------- ---------- ----------
2020/05/14 11:36:36 no data
2020/05/14 11:37:03.883 195.58 W
...  
2020/05/14 11:50:31.887 22.224 W
2020/05/14 11:50:24 22.2241 W
```

The data is shown per node if the job uses several nodes.

Further details of the options to dumpmonitoringdata for job metrics can be seen by running `help dumpmonitoringdata` within jobs mode.

**The statistics Command**

The `statistics` command shows basic statistics for historical job information. It allows statistics to be filtered per user or user group, and workload manager. The statistics can be grouped by hour, day, week or a custom interval.

**Example**

```
[bright92->wlm[slurm]->jobs]% statistics
Pending Running Finished Error Nodes
---------- ---------- ---------- ---------- ----------
0 0 31 0 31
```

Further details of the options to statistics can be seen by running `help statistics`.
15

Monitoring: Job Accounting

15.1 Introduction

In addition to the concept of metrics for devices (Chapter 13), or the concept of metrics for jobs (Chapter 14), there is also the concept of metrics for aggregations, for resources used during jobs. This last one is typically metrics aggregated per user, for resources used during jobs. Aggregation-based metrics for jobs is more conveniently called job accounting, since it resembles the idea of an accountant watching over users to track their resource use while they carry out their jobs.

The concept and implementation of aggregation-based metrics for jobs, or job accounting, are described in this Chapter.

Typically in the cluster manager, for jobs, resource usage can be presented per user. For example, if there are jobs in a queue that are being processed, the jobs can be listed:

Example

```
[bright92->wlm[slurm]->jobs]% list | head
Type Job ID User Queue Running time Status Nodes
------------ ------------ ------ ------- ------------ ---------- -------------------
Slurm 1325 tim defq 1m 2s COMPLETED node001..node003
Slurm 1326 tim defq 1m 1s COMPLETED node001..node003
Slurm 1327 tim defq 1m 2s COMPLETED node001..node003
Slurm 1328 tim defq 32s RUNNING node001..node003
Slurm 1329 tim defq 0s PENDING
```

The resource usage statistics gathered per user, for example a user tim, can then be analyzed and visualized using the job accounting interface of Bright View (section 15.5).

15.2 Labeled Entities

In job accounting, job metrics during a run are tagged with extra labels, such as the job ID, host name, and the user running the job. The modified job metrics object that is tagged in this way then becomes a job accounting-related object, called a labeled entity.

Administrators interested in using job accounting can simply skip ahead and start reading about the Bright View job accounting interface in 15.5, and just explore it directly. Those who would prefer some background on how job accounting is integrated with the cluster manager and PromQL, can continue reading this section (15.2) and the next one (15.3).

15.2.1 Dataproducers For Labeled Entities

To view labeled entities in cmsh, the path to the labeledentity submode is:

```
cmsh→monitoring→labeledentity
```

The labeledentity submode allows job accounting-related objects, called labeled entities, to be viewed. The labels are in the form `<key>"<value>"`, for example: `hostname="node001", or user="alice"`. 

The default, existing labeled entities are created from the built-in JobSampler and JobMetadataSampler dataproducers when a job is run. Custom samplers, of type prometheus, can be used to create further custom labeled entities. A custom sampler dataproducer, for example customsamplerextras, can be created from the monitoring setup mode of cmsh as follows:

Example

```
[bright92]% monitoring setup
[bright92->monitoring->setup]% add prometheus customsamplerextras
[bright92->monitoring->setup*customsamplerextras*]%
```

The customsamplerextras dataproducer can now have its properties configured and committed as described in section 13.5.4.

15.2.2 PromQL And Labeled Entities

Labeled entities can be used by administrators to help create and debug job-related queries in the Prometheus query language, PromQL. PromQL is a part of the Prometheus monitoring and alerting toolkit (https://prometheus.io). Basic PromQL documentation is available at https://prometheus.io/docs/prometheus/latest/querying/basics/.

15.2.3 Job IDs And Labeled Entities

Each job ID has a number of labeled entities associated with it. Since the number of labeled entities scales with the number of nodes and jobs, the number of labeled entities can be very large. Therefore, if examining these entities using the CMDaemon front ends such as cmsh or Bright View, then filtering or sorting the output is useful. For example, labeled entities associated with node001, and with the JobSampler data producer, and with job 1329 from the preceding output, could be viewed by filtering the full list of labeled entities as follows (output truncated and ellipsized):

Example

```
[bright92->monitoring->labeledentity]% list|head -2; list|grep 'job_id="1329"' |grep node001 |grep JobSampler
```

```
Index Name (key) ...
------ --------------------------------------------------------------------------------------...
45446 hostname="node001",job="JobSampler",job_id="1329",wlm="slurm" ...
45447 device="vda",hostname="node001",job="JobSampler",job_id="1329",wlm="slurm" ...
45448 device="vda",hostname="node001",job="JobSampler",job_id="1329",mode="read",wlm="slurm"...
```

15.2.4 Measurables And Labeled Entities

The measurables (metrics) for an entity can be listed with the measurables (metrics) command. For a particular entity with a JobSampler property and index value of 45447, the command can be run as follows:

Example

```
[bright92->monitoring->labeledentity]% measurables 45447
```

```
Type Name Parameter Class Producer
------- ---------------------------------- -------------- ----------- --------------
Metric job_blkio_sectors Prometheus JobSampler
Metric job_blkio_time_seconds Prometheus JobSampler
```

In the labeledentity mode of cmsh, the measurables listing command, which lists the measurables for labeled entities, should not be confused with the measurable navigation command, which brings the administrator to the measurable submode under the main monitoring mode.
15.3 PromQL Queries

15.3.1 The Default PromQL Queries...

By default there are several predefined PromQL queries already available. The queries can be listed from the query submode:

Example

```
[monitoring->query] $ list
```

<table>
<thead>
<tr>
<th>Name (key)</th>
<th>Start time</th>
<th>End time</th>
<th>Interval</th>
<th>Class</th>
</tr>
</thead>
<tbody>
<tr>
<td>account_job_effective_cpu_seconds</td>
<td>now</td>
<td>0s</td>
<td></td>
<td>accounting</td>
</tr>
<tr>
<td>account_job_io_bytes</td>
<td>now</td>
<td>0s</td>
<td></td>
<td>accounting</td>
</tr>
<tr>
<td>account_job_memory_usage_bytes</td>
<td>now</td>
<td>0s</td>
<td></td>
<td>accounting</td>
</tr>
<tr>
<td>account_job_running_count</td>
<td>now</td>
<td>0s</td>
<td></td>
<td>accounting</td>
</tr>
<tr>
<td>account_job_waiting_seconds</td>
<td>now</td>
<td>0s</td>
<td></td>
<td>accounting</td>
</tr>
<tr>
<td>account_job_wall_clock_seconds</td>
<td>now</td>
<td>0s</td>
<td></td>
<td>accounting</td>
</tr>
<tr>
<td>account_job_wasted_cpu_seconds</td>
<td>now</td>
<td>0s</td>
<td></td>
<td>accounting</td>
</tr>
<tr>
<td>accounts_wasted_memory</td>
<td>now</td>
<td>0s</td>
<td></td>
<td>accounting</td>
</tr>
<tr>
<td>cluster_cpu_usage_percent</td>
<td>now-1d</td>
<td>now</td>
<td>15m</td>
<td>cluster</td>
</tr>
<tr>
<td>cpu_usage_by_cluster</td>
<td>now-1d</td>
<td>now</td>
<td>1h</td>
<td>kubernetes</td>
</tr>
<tr>
<td>cpu_usage_by_deployment</td>
<td>now-1d</td>
<td>now</td>
<td>1h</td>
<td>kubernetes</td>
</tr>
<tr>
<td>cpu_usage_by_namespace</td>
<td>now-1d</td>
<td>now</td>
<td>1h</td>
<td>kubernetes</td>
</tr>
<tr>
<td>fs_usage_by_cluster</td>
<td>now-1d</td>
<td>now</td>
<td>1h</td>
<td>kubernetes</td>
</tr>
<tr>
<td>fs_usage_by_deployment</td>
<td>now-1d</td>
<td>now</td>
<td>1h</td>
<td>kubernetes</td>
</tr>
<tr>
<td>fs_usage_by_namespace</td>
<td>now-1d</td>
<td>now</td>
<td>1h</td>
<td>kubernetes</td>
</tr>
<tr>
<td>job_effective_cpu_seconds_job_name_for_user</td>
<td>now</td>
<td>0s</td>
<td></td>
<td>accounting/level/1</td>
</tr>
<tr>
<td>job_information_by_account</td>
<td>now</td>
<td>0s</td>
<td></td>
<td>drilldown/level/0</td>
</tr>
<tr>
<td>job_information_by_job_id_for_account_and_user_and_job_name</td>
<td>now</td>
<td>0s</td>
<td>drilldown/level/3</td>
<td></td>
</tr>
<tr>
<td>job_information_by_job_id_for_user</td>
<td>now</td>
<td>0s</td>
<td></td>
<td>drilldown/level/1</td>
</tr>
<tr>
<td>job_information_by_job_name_for_account</td>
<td>now</td>
<td>0s</td>
<td></td>
<td>drilldown/level/2</td>
</tr>
<tr>
<td>job_information_by_job_name_for_account_and_user</td>
<td>now</td>
<td>0s</td>
<td></td>
<td>drilldown/level/2</td>
</tr>
<tr>
<td>job_information_by_job_name_for_user</td>
<td>now</td>
<td>0s</td>
<td></td>
<td>drilldown/level/1</td>
</tr>
<tr>
<td>job_information_by_user</td>
<td>now</td>
<td>0s</td>
<td></td>
<td>drilldown/level/0</td>
</tr>
<tr>
<td>job_information_by_user_for_account</td>
<td>now</td>
<td>0s</td>
<td></td>
<td>drilldown/level/1</td>
</tr>
<tr>
<td>job_information_by_user_for_account_and_job_name</td>
<td>now</td>
<td>0s</td>
<td></td>
<td>drilldown/level/2</td>
</tr>
<tr>
<td>job_io_bytes_per_job_name_for_user</td>
<td>now</td>
<td>0s</td>
<td></td>
<td>accounting/level/1</td>
</tr>
<tr>
<td>job_memory_usage_bytes_per_job_name_for_user</td>
<td>now</td>
<td>0s</td>
<td></td>
<td>accounting/level/1</td>
</tr>
<tr>
<td>job_running_count_job_name_for_user</td>
<td>now</td>
<td>0s</td>
<td></td>
<td>accounting/level/1</td>
</tr>
<tr>
<td>job_waiting_seconds_job_name_for_user</td>
<td>now</td>
<td>0s</td>
<td></td>
<td>accounting/level/1</td>
</tr>
<tr>
<td>job_wall_clock_seconds_job_name_for_user</td>
<td>now</td>
<td>0s</td>
<td></td>
<td>accounting/level/1</td>
</tr>
<tr>
<td>job_wasted_cpu_seconds_job_name_for_user</td>
<td>now</td>
<td>0s</td>
<td></td>
<td>accounting/level/1</td>
</tr>
<tr>
<td>jobs_wasted_allocated_gpus</td>
<td>now</td>
<td>0s</td>
<td></td>
<td>accounting</td>
</tr>
<tr>
<td>memory_usage_by_cluster</td>
<td>now-1d</td>
<td>now</td>
<td>1h</td>
<td>kubernetes</td>
</tr>
<tr>
<td>memory_usage_by_deployment</td>
<td>now-1d</td>
<td>now</td>
<td>1h</td>
<td>kubernetes</td>
</tr>
<tr>
<td>memory_usage_by_namespace</td>
<td>now-1d</td>
<td>now</td>
<td>1h</td>
<td>kubernetes</td>
</tr>
<tr>
<td>net_usage_by_cluster</td>
<td>now-1d</td>
<td>now</td>
<td>1h</td>
<td>kubernetes</td>
</tr>
<tr>
<td>net_usage_by_deployment</td>
<td>now-1d</td>
<td>now</td>
<td>1h</td>
<td>kubernetes</td>
</tr>
<tr>
<td>net_usage_by_namespace</td>
<td>now-1d</td>
<td>now</td>
<td>1h</td>
<td>kubernetes</td>
</tr>
<tr>
<td>storage_ceph_host_apply_latency</td>
<td>now-1d</td>
<td>now</td>
<td>15m</td>
<td>ceph</td>
</tr>
<tr>
<td>storage_ceph_host_commit_latency</td>
<td>now-1d</td>
<td>now</td>
<td>15m</td>
<td>ceph</td>
</tr>
<tr>
<td>storage_ceph_host_cpu_usage</td>
<td>now-1d</td>
<td>now</td>
<td>15m</td>
<td>ceph</td>
</tr>
<tr>
<td>storage_ceph_osd_bytes_used</td>
<td>now-1d</td>
<td>now</td>
<td>15m</td>
<td>ceph</td>
</tr>
<tr>
<td>storage_ceph_osd_op_ratio</td>
<td>now-1d</td>
<td>now</td>
<td>15m</td>
<td>ceph</td>
</tr>
<tr>
<td>unused_gpu_job_name_for_user</td>
<td>now</td>
<td>0s</td>
<td></td>
<td>accounting/level/1</td>
</tr>
</tbody>
</table>
The queries can be conceptually divided into their classes, which at the time of writing (November 2019) are: accounting, ceph, cluster, jobs, kubernetes, along with various drilldown levels which aggregates the queries according to various groups. The grouping for drilldown levels can be confusing, and the drilldownoverview command (section 15.8.1) can be helpful in clarifying the query intention for these.

By default, queries of all classes are sampled over a period, except for the accounting and drilldown metrics.

A metric in the accounting class query is evaluated (interpolated) from existing values. These existing values are raw samples gathered over the period, up to the time when the query is evaluated.

15.3.2 ...And A Short Description Of Them
The description of each query can be listed with a little cmsh and unix text utility juggling:

Example

```bash
[bright92->monitoring->query] $ foreach * (get name; get description) | paste - - | expand -t 60
```

This yields the following table:

<table>
<thead>
<tr>
<th>Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>wasted_memory.job_name_for_user</td>
<td>The sum of the minimal wasted memory over all nodes by job_name for a user for the last period</td>
</tr>
</tbody>
</table>

...continues
### Table 15.3: PromQL Query Descriptions...continued

<table>
<thead>
<tr>
<th>Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>users_wasted_memory</code></td>
<td>The sum of the minimal wasted memory over all nodes per user for the last period</td>
</tr>
<tr>
<td><code>job_wall_clock_seconds_job_name_for_user</code></td>
<td>Wall clock time used by <code>job_name</code> for a user for the last period</td>
</tr>
<tr>
<td><code>users_job_wall_clock_seconds</code></td>
<td>Wall clock time used by users for the last period</td>
</tr>
<tr>
<td><code>job_wasted_cpu_seconds_job_name_for_user</code></td>
<td>CPU seconds allocated but not used by <code>job_name</code> for a user for the last period</td>
</tr>
<tr>
<td><code>users_job_wasted_cpu_seconds</code></td>
<td>CPU seconds allocated but not used by users for the last period</td>
</tr>
<tr>
<td><code>job_effective_cpu_seconds_job_name_for_user</code></td>
<td>CPU seconds effectively used by <code>job_name</code> for a user for the last period</td>
</tr>
<tr>
<td><code>users_job_effective_cpu_seconds</code></td>
<td>CPU seconds effectively used by users for the last period</td>
</tr>
<tr>
<td><code>job_running_count_job_name_for_user</code></td>
<td>Number of jobs running by <code>job_name</code> for a user during the last period</td>
</tr>
<tr>
<td><code>users_job_running_count</code></td>
<td>Number of jobs running by users during the last period</td>
</tr>
<tr>
<td><code>job_waiting_seconds_job_name_for_user</code></td>
<td>Total waiting time for jobs by <code>job_name</code> for a user in seconds during the last period</td>
</tr>
<tr>
<td><code>users_job_waiting_seconds</code></td>
<td>Total waiting time for users jobs in seconds during the last period</td>
</tr>
<tr>
<td><code>job_memory_usage_bytes_per_job_name_for_user</code></td>
<td>Total memory usage by <code>job_name</code> for a user during the last period in Bytes</td>
</tr>
<tr>
<td><code>users_job_memory_usage_bytes</code></td>
<td>Total memory usage by users during the last period in Bytes</td>
</tr>
<tr>
<td><code>job_io_bytes_per_job_name_for_user</code></td>
<td>Total I/O by <code>job_name</code> for a user during the last period in Bytes</td>
</tr>
<tr>
<td><code>users_job_io_bytes</code></td>
<td>Total I/O by users during the last period in Bytes</td>
</tr>
<tr>
<td><code>wasted_memory_job_name_for_account</code></td>
<td>The sum of the minimal wasted memory over all nodes by <code>job_name</code> for a account for the last period</td>
</tr>
<tr>
<td><code>accounts_wasted_memory</code></td>
<td>The sum of the minimal wasted memory over all nodes per account for the last period</td>
</tr>
</tbody>
</table>

...continues
Table 15.3: PromQL Query Descriptions...continued

<table>
<thead>
<tr>
<th>Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>account_job_wall_clock_seconds</td>
<td>Wall clock time used by account for the last period</td>
</tr>
<tr>
<td>account_job_wasted_cpu_seconds</td>
<td>CPU seconds allocated but not used by account for the last period</td>
</tr>
<tr>
<td>account_job_effective_cpu_seconds</td>
<td>CPU seconds effectively used by account for the last period</td>
</tr>
<tr>
<td>account_job_running_count</td>
<td>Number of jobs running by account during the last period</td>
</tr>
<tr>
<td>account_job_waiting_seconds</td>
<td>Total waiting time for account jobs in seconds during the last period</td>
</tr>
<tr>
<td>account_job_memory_usage_bytes</td>
<td>Total memory usage by account during the last period in Byte seconds</td>
</tr>
<tr>
<td>users_job_allocated_nodes</td>
<td>Number of nodes allocated by users</td>
</tr>
<tr>
<td>users_job_cpu_usage</td>
<td>Effective CPU usage by users</td>
</tr>
<tr>
<td>users_job_waiting</td>
<td>Number of jobs currently waiting for every user</td>
</tr>
<tr>
<td>users_job_memory_bytes</td>
<td>Current memory consumption for user jobs in Bytes</td>
</tr>
<tr>
<td>users_job_io_bytes_per_second</td>
<td>Current I/O for user jobs in B/s</td>
</tr>
<tr>
<td>cluster_cpu_usage_percent</td>
<td>CPU usage percentage over all nodes up</td>
</tr>
<tr>
<td>storage_ceph_osd_bytes_used</td>
<td>The amount of disk space used across all OSDs</td>
</tr>
<tr>
<td>storage_ceph_host_apply_latency</td>
<td>The average apply latency on each host running Ceph OSDs</td>
</tr>
<tr>
<td>storage_ceph_host_commit_latency</td>
<td>The average commit latency on each host running Ceph OSDs</td>
</tr>
<tr>
<td>storage_ceph_host_cpu_usage</td>
<td>The cpu usage on each host running Ceph OSDs</td>
</tr>
<tr>
<td>storage_ceph_osd_op_ratio</td>
<td>The number of operations per second on each Ceph OSD</td>
</tr>
<tr>
<td>memory_usage_by_deployment</td>
<td>Total memory usage by deployment during the last week in Bytes per second</td>
</tr>
<tr>
<td>memory_usage_by_cluster</td>
<td>Total memory usage by cluster during the last week in Bytes per second</td>
</tr>
<tr>
<td>memory_usage_by_namespace</td>
<td>Total memory usage by namespace during the last week in Bytes per second</td>
</tr>
<tr>
<td>Name</td>
<td>Description</td>
</tr>
<tr>
<td>-------------------------------------------</td>
<td>-------------------------------------------------------</td>
</tr>
<tr>
<td>fs_usage_by_deployment</td>
<td>Current FS I/O by deployment in Bytes</td>
</tr>
<tr>
<td>fs_usage_by_cluster</td>
<td>Current FS I/O by cluster in Bytes</td>
</tr>
<tr>
<td>fs_usage_by_namespace</td>
<td>Current FS I/O by namespace in Bytes</td>
</tr>
<tr>
<td>net_usage_by_deployment</td>
<td>Network usage by deployment in Bytes per second</td>
</tr>
<tr>
<td>net_usage_by_cluster</td>
<td>Network usage by cluster in Bytes per second</td>
</tr>
<tr>
<td>net_usage_by_namespace</td>
<td>Network usage by namespace in Bytes per second</td>
</tr>
<tr>
<td>cpu_usage_by_deployment</td>
<td>CPU usage by deployment in nr. of cores</td>
</tr>
<tr>
<td>cpu_usage_by_cluster</td>
<td>CPU usage by cluster in nr. of cores</td>
</tr>
<tr>
<td>cpu_usage_by_namespace</td>
<td>CPU usage by namespace in nr. of cores</td>
</tr>
<tr>
<td>job_information_by_job_id_for_account_and_user_and_job_name</td>
<td>Generic job information drill down query grouped by wlm and job_id for a specific account, user and job_name</td>
</tr>
<tr>
<td>job_information_by_job_name_for_account_and_user</td>
<td>Generic job information drill down query grouped by job_name for a specific account and user</td>
</tr>
<tr>
<td>job_information_by_user_for_account_and_job_name</td>
<td>Generic job information drill down query grouped by user for a specific account and job_name</td>
</tr>
<tr>
<td>job_information_by_job_id_for_user_and_job_name</td>
<td>Generic job information drill down query grouped by wlm and job_id for a specific user and job_name</td>
</tr>
<tr>
<td>job_information_by_user_for_account</td>
<td>Generic job information drill down query grouped by user for a specific account</td>
</tr>
<tr>
<td>job_information_by_job_name_for_account</td>
<td>Generic job information drill down query grouped by job_name for a specific account</td>
</tr>
<tr>
<td>job_information_by_job_name_for_user</td>
<td>Generic job information drill down query grouped by job_name for a specific user</td>
</tr>
<tr>
<td>job_information_by_job_id_for_user</td>
<td>Generic job information drill down query grouped by wlm and job_id for a specific user</td>
</tr>
<tr>
<td>job_information_by_account</td>
<td>Generic job information drill down query grouped by account</td>
</tr>
<tr>
<td>job_information_by_user</td>
<td>Generic job information drill down query grouped by user</td>
</tr>
<tr>
<td>jobs_wasted_allocated_gpus</td>
<td>Average % of allocated GPUs wasted for jobs that ran in the specified period, averaged and grouped by job_id</td>
</tr>
</tbody>
</table>
### Table 15.3: PromQL Query Descriptions...continued

<table>
<thead>
<tr>
<th>Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>users_wasted_allocated_gpus</code></td>
<td>Average % of allocated GPUs wasted for jobs that ran in the specified period, averaged and grouped by user</td>
</tr>
<tr>
<td><code>wasted_allocated_gpus_for_user</code></td>
<td>Average % of allocated GPUs wasted for jobs that ran in the specified period, averaged and grouped by <code>job_id</code>, for a particular user</td>
</tr>
<tr>
<td><code>used_gpu_job_name_for_user</code></td>
<td>Used GPUs, for values of use greater than or equal to 0.1%, averaged and grouped by job names that ran on them in the specified period, for a particular user</td>
</tr>
<tr>
<td><code>users_used_gpu</code></td>
<td>Used GPUs, for values of use greater than or equal to 0.1%, averaged and grouped by users using them in the specified period</td>
</tr>
<tr>
<td><code>unused_gpu_job_name_for_user</code></td>
<td>Unused GPUs, for values of use less than 0.1%, averaged and grouped by job names using them in the specified period, for a particular user</td>
</tr>
<tr>
<td><code>users_unused_gpu</code></td>
<td>Unused GPUs, for values of use less than 0.1%, averaged and grouped by users using them in the specified period</td>
</tr>
</tbody>
</table>

The listings give an idea of what the query does.

For example, for the `users_job_cpu_usage` utility, the idea is that it shows the CPU usage for jobs for each user.

#### 15.3.3 Modifying The Default PromQL Query Properties

The properties of a particular query can be shown and modified:

**Example**

```
[bright92->monitoring->query]% use users_job_cpu_usage
[bright92->monitoring->query[users_job_cpu_usage]]% show
```

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>Name</td>
<td>users_job_cpu_usage</td>
</tr>
<tr>
<td>Revision</td>
<td>jobs</td>
</tr>
<tr>
<td>Class</td>
<td>jobs</td>
</tr>
<tr>
<td>Alias</td>
<td></td>
</tr>
<tr>
<td>Start time</td>
<td>now-1d</td>
</tr>
<tr>
<td>End time</td>
<td>now</td>
</tr>
<tr>
<td>Interval</td>
<td>15m</td>
</tr>
<tr>
<td>Description</td>
<td>Effective CPU usage by users</td>
</tr>
<tr>
<td>PromQL Query</td>
<td>&lt;136B&gt;</td>
</tr>
<tr>
<td>Access</td>
<td>Public</td>
</tr>
<tr>
<td>Unit</td>
<td>CPU</td>
</tr>
<tr>
<td>Price</td>
<td>0.000000</td>
</tr>
<tr>
<td>Currency</td>
<td>$</td>
</tr>
<tr>
<td>Preference</td>
<td>0</td>
</tr>
<tr>
<td>Drill down</td>
<td>&lt;0 in submode&gt;</td>
</tr>
<tr>
<td>Notes</td>
<td>&lt;0B&gt;</td>
</tr>
</tbody>
</table>

The PromQL query code itself is typically a few lines long, and can also be viewed and modified using `get`, and `set`. 
An Example PromQL Query, Properties, And Disassembly

The `users_job_cpu_usage` query is a standard predefined query, and is used as an example here. The query shows the CPU usage by a user around the time the sample was taken. It is sometimes called an “instantaneous” value. However it is not that instantaneous, because its value is calculated by taking samples of the CPU usage over the last 10 minutes of the job run rather than at the query time. The code for the query can be viewed with:

```
[bright92->monitoring->query]% get users_job_cpu_usage promqlquery
sum by(user) (irate(job_cpuacct_usage_seconds[10m]) * on(job_id, hostname) group_right())
```

For those unfamiliar with PromQL, some disassembly of the `users_job_cpu_usage` query is helpful. Terminology used by PromQL and the cluster manager, for the pieces used to build the query, is listed in the following table:

<table>
<thead>
<tr>
<th>PromQL Terminology</th>
<th>Example</th>
<th>the cluster manager Terminology</th>
</tr>
</thead>
<tbody>
<tr>
<td>Query</td>
<td><code>users_job_cpu_usage</code></td>
<td>PromQL query</td>
</tr>
<tr>
<td>Instant query</td>
<td><code>job_cpuacct_usage_seconds</code></td>
<td>Metric (from JobSampler dataproducer, belonging to the Prometheus class)</td>
</tr>
<tr>
<td>Range vector</td>
<td><code>job_cpuacct_usage_seconds[10m]</code></td>
<td>Metric samples over a time span</td>
</tr>
</tbody>
</table>

As was mentioned before: job account metrics, unlike traditional metrics, are not directly associated with the device-related objects. For such metrics, the monitoring data command `dumpmonitoringdata` is therefore not accessed in `cmsh` from `device` mode or `category` mode.

Instead, the Prometheus metric `job_cpuacct_usage_seconds`, for example, is accessed via the `labeledentities` mode.

The properties and interpolated values at a particular instant of time for the metric can be accessed via an `instantquery` such as (some output excised for clarity):

```
[bright92->monitoring->labeledentity]% instantquery job_cpuacct_usage_seconds
__name__ hostname job job_id uid user wlm Timestamp Value
-------------------------- -------- ----------- ------- ---- ----- ----- --------- ----- 
job_cpuacct_usage_seconds node001 JobSampler 623 1002 tony slurm 15:52:15 129
job_cpuacct_usage_seconds node002 JobSampler 624 1002 tony slurm 15:52:15 71
...
```

The properties and interpolated values of the metric over a range of time can be accessed via a `rangequery` such as (some output excised for clarity):

```
[bright92->monitoring->labeledentity]% rangequery --start now-2h --end now job_cpuacct_usage_seconds
__name__ hostname job job_id uid user wlm Timestamp Value
-------------------------- -------- ----------- ------- ---- ----- ----- -------- ----- 
job_cpuacct_usage_seconds node001 JobSampler 623 1002 tony slurm 13:52:15 82
job_cpuacct_usage_seconds node001 JobSampler 623 1002 tony slurm 14:52:15 97
```
The association of the `instantquery` and `rangequery` output with job accounts is because its data-producer is `JobSampler`.

Further options for the `instantquery` and `rangequery` commands can be found in their help texts within `cmsh`.

15.3.5 Aside: Getting Raw Values For A Prometheus Class Metric

The PromQL language is aimed at providing an overall view of jobs and resource usage. The actual individual raw values that Prometheus metrics are built on—the entries in the Time Series Database (TSDB)—are not regarded as being important for the end user. The emphasis in PromQL is on seeing the values as seen by statistical reworking.

This section, which is about the raw TSDB values, is thus provided as background information for administrators who would anyway like to see what the raw values look like.

Raw values of the metric for a job ID can be accessed by using the index of the labeled identity that is associated with that job ID. For example, job ID 624 can have its index found with some grepping (some output elided or excised for clarity):

**Example**

```bash
[bright92->monitoring->labeledentity]% list|head -2; list|grep 'hostname='| grep 'job_id="624"'
Index Name (key) Introduction Last used
------ ----------------------------------------------------- ------------ ---------
4060 hostname="node001",job="JobSampler",job_id="624", ... 13:30:03 16:00:03
```

The index for the job ID 624 is 4060. The job ID can be used by the `dumpmonitoringdata` command to show the series raw values along with their time stamps:

**Example**

```bash
[bright92->monitoring->labeledentity]% dumpmonitoringdata -24h now job_cpuacct_usage_seconds 4060
```

<table>
<thead>
<tr>
<th>Timestamp</th>
<th>Value</th>
<th>Info</th>
</tr>
</thead>
<tbody>
<tr>
<td>2019/08/01 16:08:03.255</td>
<td>10s</td>
<td></td>
</tr>
<tr>
<td>2019/08/01 16:10:03.255</td>
<td>2m 10s</td>
<td></td>
</tr>
<tr>
<td>2019/08/01 16:12:03.255</td>
<td>4m 9s</td>
<td></td>
</tr>
<tr>
<td>2019/08/01 16:14:03.255</td>
<td>6m 9s</td>
<td></td>
</tr>
<tr>
<td>2019/08/01 16:16:03.255</td>
<td>8m 8s</td>
<td></td>
</tr>
<tr>
<td>2019/08/01 16:18:03.255</td>
<td>10m 8s</td>
<td></td>
</tr>
<tr>
<td>2019/08/01 16:20:03.255</td>
<td>12m 7s</td>
<td></td>
</tr>
<tr>
<td>2019/08/01 16:22:03.255</td>
<td>14m 6s</td>
<td></td>
</tr>
<tr>
<td>2019/08/01 16:24:03.255</td>
<td>16m 6s</td>
<td></td>
</tr>
<tr>
<td>2019/08/01 16:26:03.255</td>
<td>18m 5s</td>
<td></td>
</tr>
<tr>
<td>2019/08/01 16:28:03.255</td>
<td>20m 5s</td>
<td></td>
</tr>
<tr>
<td>2019/08/01 16:30:03.255</td>
<td>no data</td>
<td></td>
</tr>
</tbody>
</table>

These raw values are the values that are used for interpolation during PromQL queries.

The label names for job samples can be seen using the index:
15.3 PromQL Queries

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>Index</td>
<td>4060</td>
</tr>
<tr>
<td>Introduction</td>
<td>Thu, 01 Aug 2019 16:08:03 CEST</td>
</tr>
<tr>
<td>Last used</td>
<td>Thu, 01 Aug 2019 16:30:03 CEST</td>
</tr>
<tr>
<td>Name</td>
<td>hostname=&quot;node001&quot;, job=&quot;JobSampler&quot;, job_id=&quot;624&quot;, uid=&quot;1002&quot;, user=&quot;tony&quot;, wlm=&quot;slurm&quot;</td>
</tr>
<tr>
<td>Permanent</td>
<td>no</td>
</tr>
<tr>
<td>Revision</td>
<td></td>
</tr>
</tbody>
</table>

15.3.6 ...An Example PromQL Query, Properties, And Disassembly (Continued)

Getting back from the aside about raw values, and continuing on with the example PromQL query from the start of this section (page 677), the query code for the query `users_job_cpu_usage` was:

```promql
sum by(user) (
  irate(job_cpuacct_usage_seconds[10m]) * on(job_id, hostname) group_right()
  (job_metadata_is_running)
)
```

With the necessary background explanations having been carried out, the disassembly of this query can now be done:

The core of the query is built around the job sampler metric `job_cpuacct_usage_seconds`.

The `irate` measurement in this case calculates the rate of change based on the last two most recent values in the Prometheus range vector.

The Prometheus range vector is formed from the Prometheus instant query by using the square brackets with a time value enclosed (`job_cpuacct_usage_seconds[10m]`). The Prometheus instant query is, as the terminology table earlier pointed out, the Prometheus version of the job sampler metric.

Getting back to the range vector, a range vector in general is a series of values formed from the corresponding Prometheus instant query. With the instant query being `job_cpuacct_usage_seconds` here, the range vector is formed over a span of 10 minutes.

After the `irate` function has taken the average, the resultant is what in PromQL is called an instant vector value. This consists of data in the form [CPU seconds consumed during period, timestamp associated with time period sample]. The instant vector value is then joined against each vector element of the pair `job_id`, `hostname` to generate the labeled identifier for the job running on the node. The `group_right` of the result uses the `job_id` and `hostname` as the leading labels in the label identifiers.

The `job_metadata_is_running` function means that values are generated only while `job_metadata` is running. The `sum by(user)` function means that sum of the result is aggregated per user.

Visualisation based on the result is most easily carried out by plotting job CPU usage for each user against time in the period specified, which can be done in a more user-friendly way with Bright View.

The `users_job_wall_clock_seconds` query, is similar, and can be used to plot wall clock seconds consumed by a user over the last period:

Example

```promql
max_over_time(job_metadata_running_seconds[$period]) * on(job_id) group_right()
max_over_time(job_metadata_num_cpus[$period])
```
Predefined queries can be executed in the labeledentity mode with the -q option:

\[
\text{[bright92->monitoring->labeledentity]% instantquery -q users_job_wall_clock_seconds}
\]

<table>
<thead>
<tr>
<th>user</th>
<th>Timestamp</th>
<th>Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>alice</td>
<td>Mon Jun 24 10:19:34 2019</td>
<td>28.787</td>
</tr>
<tr>
<td>charline</td>
<td>Mon Jun 24 10:19:34 2019</td>
<td>102.83</td>
</tr>
<tr>
<td>eve</td>
<td>Mon Jun 24 10:19:34 2019</td>
<td>58.574</td>
</tr>
<tr>
<td>frank</td>
<td>Mon Jun 24 10:19:34 2019</td>
<td>59.575</td>
</tr>
</tbody>
</table>

If Bright View is used instead of cmsh, then Prometheus queries can be accessed via the clickpath:

\[
\text{Monitoring → PromQL Queries}
\]

### 15.4 Parameterized PromQL Queries

It is also possible to create parameterized queries using the \(<\text{key}>=\<\text{value}\>\) labels in the labeled entities. This is handy for running the same query with different parameters.

For example, an existing unparameterized query

\[
\text{users_job_wall_clock_seconds}
\]

can be used as a starting point:

\[
\text{[bright92->monitoring->query]% get users_job_wall_clock_seconds promqlquery}
\]

\[
\text{sum by(user) (}
\max_{\text{over time}}(\text{job_metadata_running_seconds}\{\text{account}="\$\{account\}"\}\{\$period\})
\times \text{on(job_id) group_right()}
\max_{\text{over time}}(\text{job_metadata_num_cpus}\{\text{account}="\$\{account\}"\}\{\$period\})
\text{)}
\]

The original query can be cloned over to a new, soon-to-be-parameterized, query called

\[
\text{users_job_wall_clock_per_account_seconds}
\]

can be used for convenient development:

\[
\text{[bright92->monitoring->query]% clone users_job_wall_clock_seconds promqlquery users_job_wall}
\text{l_clock_per_account_seconds promqlquery}
\]

One or more \(\$\{\text{parameter}\}\) fields can then be added to the query. All fields in

\[
\text{users_job_wall_clock_per_account_seconds}
\]

can be replaced verbatim. So any part of the query can be made into a parameter.

\[
\text{[bright92->monitoring->query]% get users_job_wall_clock_per_account_seconds promqlquery}
\]

\[
\text{sum by(user) (}
\max_{\text{over time}}(\text{job_metadata_running_seconds}\{\text{account}="\$\{account\}"\}\{\$period\})
\times \text{on(job_id) group_right()}
\max_{\text{over time}}(\text{job_metadata_num_cpus}\{\text{account}="\$\{account\}"\}\{\$period\})
\text{)}
\]

The new query is then committed and can be run by the administrator on demand.

The original query sums over all accounts:

\[
\text{[bright92->monitoring->labeledentity]% instantquery -q users_job_wall_clock_seconds}
\]

<table>
<thead>
<tr>
<th>user</th>
<th>Timestamp</th>
<th>Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>charline</td>
<td>Mon Jun 24 10:25:20 2019</td>
<td>102.83</td>
</tr>
<tr>
<td>eve</td>
<td>Mon Jun 24 10:25:20 2019</td>
<td>58.574</td>
</tr>
<tr>
<td>frank</td>
<td>Mon Jun 24 10:25:20 2019</td>
<td>85.362</td>
</tr>
</tbody>
</table>
The parameterized query lets the administrator run the same query for specific accounts. For example, if there are slurm accounts for physics phys and mathematics math, then here account=phys and account=math, are the \langle key=\rangle=\langle value=\rangle" format options. The query can then be run with the -pl--parameter option as follows:

```
[bright92->monitoring->labeledentity]% instantquery -q users_job_wall_clock_per_account_second
  s -p account=phys
```

```
user  Timestamp          Value
---------- ------------------------- --------
charline Mon Jun 24 10:25:22 2019 29.787
frank  Mon Jun 24 10:25:22 2019 30.788
```

```
[bright92->monitoring->labeledentity]% instantquery -q users_job_wall_clock_per_account_second
  s -p account=math
```

```
user  Timestamp          Value
---------- ------------------------- --------
charline Mon Jun 24 10:25:37 2019 73.049
eve    Mon Jun 24 10:25:37 2019 30.787
```

### 15.5 Job Accounting In Bright View

Job accounting can be viewed within Bright View’s accounting mode by clicking on the calculator icon at the top right hand corner of the Bright View standard display (figure 13.5). An accounting panel is then displayed (figure 15.1), within a dashboard:

![Figure 15.1: Job Accounting: Panel](image)

**15.5.1 Management And Use Of The Accounting Panel**

A new panel can be added by clicking on the \(\circ\) button in the dashboard panel. By clicking on the gear icon in the dashboard panel, the accounting panel layout can be changed to allow other arrangements of the panels, or to allow more panels, with other queries.
An accounting panel itself allows a previously-created query to be selected and executed. A PromQL query can be executed with default parameters by just clicking on the Run button.

The box to the right of the Run button shows the query name, and its description.

How to run and view PromQL queries in the Bright View accounting panel is described in this section (15.5). The query specification in Bright View is described in more detail in section 15.6.

**PromQL Queries Input In The Accounting Panel**

Predefined PromQL queries (page 671) can be viewed via the dropdown list that appears under Saved queries. By default the panel has the query `account_job_effective_cpu_seconds` selected.

- In the default Basic mode of the panel, multiple queries of the same class can be ticked per panel (figure 15.2). To tick a new class of query in the same panel, all the queries that do not match the new class must first be unticked. In the panel of figure 15.2 the class is accounting.

PromQL queries in Basic mode can only be run as instant queries. The results of such a run are then displayed in that single panel.

- In Advanced mode (section 15.6) only one query can be selected and run per panel if an instant query is run.

However, Advanced mode does allow PromQL range queries to be run, in addition to instant queries.

**PromQL Queries Run Results Display In The Accounting Panel**

Run results can be displayed using the row toggle or the plot toggle.

- The row toggle toggles displaying the result as rows of data per user, cluster, account, or other aggregate (figure 15.1)
• The plot toggle toggles displaying the result as a pie chart, or as a standard $x$-$y$ plot:

– The pie chart, or doughnut chart, is available for a PromQL Instant mode query (figure 15.3). The chart shows how much of the resource is used at that instant per user, or per cluster, or per account, or per other aggregate. How much of the resource is used can be displayed upon the chart in figures, either as the amount itself, or as a percentage of the total aggregate amount. The chart can display aggregates using a maximum of 10 slices, by default. If needed, the right amount of the smallest extra slices are grouped together as one slice, others, so that the maximum number is not exceeded.

![Doughnut Display At An Instant](image)

Figure 15.3: Job Accounting: Doughnut Display At An Instant

– The $x$-$y$ plot is available for a PromQL Range mode query (figure 15.4). The plot shows resource use versus time. The plot can be per user, cluster, account, or other aggregate. A particular user or other aggregate can be highlighted on the plot by placing the mouse pointer over the appropriate legend color icon. The time scale can be zoomed in or out with the mouse scroll bar.
15.6 PromQL Query Modes And Specification In Bright View

Clicking on the gear icon for the Accounting Panel (figure 15.1) allows the Accounting widget mode to be toggled to Advanced. Options that can then be modified include:

- the Query Type: This can be either Instant or Range.

**An Instant Query Type**

is executed at a particular instant. It outputs a single number for each entity (usually a user). The number is the interpolated data value for the PromQL query at that time. Each data value obtained by the PromQL query is typically an aggregate of the data values over a preceding time period, where the preceding time period is defined in the PromQL query.

The single number obtained by the query for each entity is typically used to summarize a value at that instant for that entity.

An example of the Instant query type is `users_job_waiting_seconds`. This provides the total waiting time, in seconds, for each user, for the jobs the users ran during the past period. The waiting time is calculated at the instant specified for the query.

If the Time parameter for the query is kept as the default value of now, then the query provides the latest value for each user.

If the Time parameter is set to a time in the past, then the query picks up the value at that time in the past, or provides an interpolated value for that time in the past, for each user.

The display can be presented in rows of user, value pairs. As a plot, the display can be presented as a pie chart, which each slice being proportional to the value for the particular user compared
with the total value for all the users. Instead of a pie chart, the visualization can instead be a ring (or doughnut) chart, which is really just a pie chart with a hole in the centre.

PromQL allows a variety of time specifications. The interface validates whatever the interface user types in, and there is a calendar widget that allows an absolute time to be specified. Some useful time specifications are:

**Example**

<table>
<thead>
<tr>
<th>Time specification</th>
<th>What time is meant</th>
</tr>
</thead>
<tbody>
<tr>
<td>now</td>
<td>at the time it is run</td>
</tr>
<tr>
<td>now-30m</td>
<td>30 minutes ago</td>
</tr>
<tr>
<td>now-1h</td>
<td>one hour ago</td>
</tr>
<tr>
<td>now-1h/h</td>
<td>an hour ago, starting at the start of that hour</td>
</tr>
<tr>
<td>now-2d</td>
<td>48 hours ago</td>
</tr>
<tr>
<td>now-0d/d</td>
<td>today’s midnight (the most recently-passed midnight)*</td>
</tr>
<tr>
<td>now-1d/d</td>
<td>yesterday’s midnight**</td>
</tr>
<tr>
<td>Thursday, July 11, 2019 17:00:00</td>
<td>an absolute time (as set by the calendar widget)</td>
</tr>
</tbody>
</table>

*The meaning can be understood by looking at how `/d` operates. It rounds off the day by truncation, which in the case of `now-0d/d` means the midnight prior to now.

**Similarly, yesterday’s midnight is the midnight immediately prior to 24 hours ago.

The time units for PromQL are ([https://prometheus.io/docs/prometheus/latest/querying/basics/#range-vector-selectors](https://prometheus.io/docs/prometheus/latest/querying/basics/#range-vector-selectors)):

- `s` - seconds
- `m` - minutes
- `h` - hour
- `d` - day
- `w` - week
- `M` - month (31 days). `M` is not a PromQL unit, so it cannot be used inside a query. But it is a handy alias in the cluster manager for an invariant time of 31 days.
- `y` - year

The - operator in Prometheus is an offset operator, used only after `now` in the time specification field. The `/<time unit>` syntax implies a start at that unit of time.

Thus if, for example, `now-1d/d` is set as the end time, then when the query runs, it picks up the values at “yesterday’s midnight.” For each user, the value is the number of seconds the job of the user was in a wait state for the week prior to yesterday’s midnight. It rounds off the day by truncation, which in the case of `now-1d/d` means the midnight prior to 24 hours ago. Varying the user means that the number of seconds varies accordingly.

The results of a run can be

- displayed in rows as values in Bright View itself (figure 15.5)
- downloaded as a CSV file
- downloaded as an Excel spreadsheet.
A Range Query Type

This query is executed over a time period. It fetches a value for each interval within that time range, and then plots these values.

An example of the Range query type is `users_job_memory_bytes`. This fetches the instantaneous memory usage for jobs over intervals, grouped per user during that interval (figure 15.5). The query submitter can set the Start Time and End Time parameters. However, very long ranges can be computationally expensive, and the graph resolution may not be ideal when zoomed in.

![Figure 15.5: Job Accounting: User Values Over A Range](image)

- the PromQL Query: Within advanced mode, the PromQL query itself can be edited.
- the Interval: When editing a Range query type via the Saved query option, the interval for a range query run can be set. Lower numbers lead to smaller intervals, hence higher resolution and higher execution times. A value of 0 means that a reasonable default interval is attempted. Choosing a interval that is less than the sampling time of the metric is a bit pointless, and tends to lead to data values that display non-smooth behaviour.

15.7 Access Control For Workload Accounting And Reporting

The ability to view jobs is controlled by four tokens defined in a user’s profile:

1. GET_JOB_TOKEN: Allows all running jobs to be seen.
2. GET_OWN_JOB_TOKEN: Allows owned running jobs to be seen.
3. GET_JOBINFO_TOKEN: Allows all cached historic and running jobs to be seen.
4. GET_OWN_JOBINFO_TOKEN: Allows owned cached historic and running jobs to be seen.

To retrieve monitoring data for the job, the token PLOT_TOKEN must also be defined for the profile.

A job is always owned by the user that runs it. Ownership of a job can also be shared with other users by defining project managers. This establishes a 2-level hierarchy, with project managers above the subordinates, who are users that are assigned to the project manager. One or more accounts, can be assigned to project managers.
15.7 Access Control For Workload Accounting And Reporting

15.7.1 Defining Project Managers Using Internal User Management

Any user can be turned into a project manager. If the cluster manager LDAP server is being used, then the project manager can be configured via cmsh.

Example

alice can be made the project manager of bob and charlie. This allows her access to the job data of her subordinates:

```
[bright92->user]% projectmanager alice
[bright92->user[alice]->projectmanager%] set users bob charlie
[bright92->user[alice]->projectmanager%] commit
```

Example

albert can be made the manager of the physics account. This gives him access to all jobs running under that account:

```
[bright92->user]% projectmanager albert
[bright92->user[albert]->projectmanager%] set accounts physics
[bright92->user[albert]->projectmanager%] commit
```

Both mechanisms, users and accounts, can be combined to provide access control.

Example

To limit access to jobs that are running under the physics account to a specific set of users:

```
[bright92->user]% projectmanager albert
[bright92->user[albert]->projectmanager%] set accounts physics
[bright92->user[albert]->projectmanager%] set users niels richard
[bright92->user[albert]->projectmanager%] set operator and
[bright92->user[albert]->projectmanager%] commit
```

15.7.2 Defining Project Managers Using External User Management

If an external user management server is used instead of the cluster manager LDAP server, then project managers cannot be defined in cmsh. Instead, a script has to be written that provides the definitions for project managers in the form of a JSON object.

The full path of the script, for example /path/to/the/script, has to be set as a value to the ProjectManagerScript parameter in cmd.conf. This is done by adding it to the AdvancedConfig directive (page 862):

Example

```
AdvancedConfig = { "ProjectManagerScript="/path/to/the/script" }
```

The directive becomes active after restarting CMDaemon.

An example of a project manager script can be found at /cm/local/apps/cmd/scripts/cm-project-managers.py. It gives users access to each other’s jobs if they share at least one group.

The easiest way to use the script is use a mapping to inform CMDaemon which of the other user’s jobs each user has access to.

Example

So if frank can access data belonging to bob and dennis, while bob can access data belonging to dennis, while dennis can only access his own data, then the project manager configurations can be set up as:
Account access control can also be included in the output of the script, by setting values for the users, accounts, and the boolean operator (and, or) options:

Example

```
{
  "alice": {
    "users": ["charline", "eve"],
    "accounts": ["math", "chem"],
    "and": True
  },
  "eve": {
    "accounts": ["chem"]
  }
}
```

After restarting CMDaemon, it automatically runs this script when committing a change for a device or data producer. It is also possible to manually trigger the script to be run on the active head, by executing:

```
[root@bright92 ~]# echo "PROJECT.MANAGERS.UPDATE" > /var/spool/cmd/eventbucket
```

The script must not take more than a few seconds to process.

**Workaround For Project Manager Script That Takes Too Long**

If the script takes longer to run, then it must be run outside of CMDaemon, and its output should be saved as a file. If the output file is located at `/path/to/the/file`, then its path can be set as an input to the `ProjectManagerFile` parameter in `cmd.conf`. This is done by adding it to the `AdvancedConfig` directive (page 862):

Example

```
AdvancedConfig = { "ProjectManagerFile="/path/to/the/file" }
```

The project manager definitions become active after CMDaemon is restarted.

### 15.8 Drilldown Queries For Workload Accounting And Reporting

Metrics can be aggregated in various ways. Common ways are by:

- **device**: A typical hardware device is a node. Each node can then have its metrics, which are CPU usage, memory, storage, and other resource use, displayed over time. Device metrics are largely covered in this chapter in the sections up to and including section 13.8.

- **job**: With this aggregation, each job that is run by a workload manager can have its metrics, which are CPU usage, memory, storage and other resource use, displayed over time. Job metrics are covered in Chapter 14.

Other ways of metrics aggregation are part of workload accounting. With workload accounting, a workload manager runs jobs, and a job metric can be aggregated and filtered by:

- **user**
• job (job ID)
• account
• job name

The aggregation types can be applied singly. However, they can also be applied at the same time, like filters. For example:

• each user could have a particular job metric aggregated

or

• a particular user could have a particular job metric

or

• a particular user could have a particular job metric aggregated for a particular job ID only

or

• a particular user could have a particular job metric aggregated for that particular job ID only for a particular account only

A cluster administrator that uses several filters to get to the “bottom” of how resources are being used, functions in a manner reminiscent of someone drilling to the bottom to find something. This type of filtering is therefore called drilldown. Each filter corresponds to a level of drilldown.

Drilldown is a bit like how in cmsh the use of the filter command within jobs mode can narrow down what is displayed:

Example

```
$ cmsh
[bright92->vlm[slurm]->jobs]% filter -n iozone -u edgar -a projecty
```

```
Job ID Job name User Queue Submit time Start time End time Nodes Exit code
------ -------- ------ ----- ----------- ---------- -------- ----- ---------
15 iozone edgar defq 14:22:05 14:50:22 15:00:24 node001 0
19 iozone edgar defq 14:26:07 15:10:25 15:20:19 node001 0
25 iozone edgar defq 14:34:57 15:30:20 15:40:39 node001 0
36 iozone edgar defq 14:43:54 16:10:15 16:19:57 node001 0
41 iozone edgar defq 14:47:24 16:19:57 16:29:42 node001 0
```

except that in cmsh the value of the job metric is not specified or shown by the filter command.

Drilldown is also rather similar in concept to how pivot tables are used in Excel spreadsheets. Pivot tables are particular selections (like the filtered choices in drilldown). The selections are applied to a great deal of raw data. A function (like the metric in drilldown) is applied to the selection, to present the information more clearly to the end viewer.

In contrast with the filter output of cmsh, in Bright View the job metric value is made visible in a table or in graphs over the period.

15.8.1 The drilldownoverview Command

The list of predefined drilldown queries of section 15.3.1 can be listed with the drilldownoverview command. This can be run from the query submode of the monitoring mode. The output, with some columns cut out for convenience, looks like:

Example

```
$ cmsh
[bright92->monitoring->query]% drilldownoverview | cut -b1-54,80-143
```
The queries have drilldown options. For example, the `job_information_by_account`:

```
[bright92->monitoring->query][job_information_by_account]% use job_information_by_account
[rima->monitoring->query][job_information_by_account][% show
Parameter Value
Name job_information_by_account
Revision drilldown/level/0
Alias Start time now
End time
Interval 0s
Description Generic job information drill down query grouped by account
PromQL Query <61B>
Access Public
Unit Price 0.000000
Currency $
Preference 0
Drill down <2 in submode>
Notes <0B>
```

```
[bright92->monitoring->query][job_information_by_account]% drilldown
[bright92->monitoring->query][job_information_by_account][% list
Name (key) Parameters Query
job_name account job_information_by_job_name_for_account
user account job_information_by_user_for_account
```
Drilldown Queries For Workload Accounting And Reporting

```
[bright02--monitoring--query[job_information_by_account]--drilldown]% use job_name
[rims--monitoring--query[job_information_by_account]--drilldown[job_name]]% show

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>Name</td>
<td>job_name</td>
</tr>
<tr>
<td>Revision</td>
<td></td>
</tr>
<tr>
<td>Parameters</td>
<td>account</td>
</tr>
<tr>
<td>Query</td>
<td>job_information_by_job_name_for_account</td>
</tr>
</tbody>
</table>
```
16

Monitoring: Job Chargeback

16.1 Introduction

16.1.1 The Word “Chargeback”

In a non-IT context, the term chargeback is commonly used with credit cards, in a situation where a cardholder disputes a payment that was made to a merchant. When the credit card company pays back the disputed payment to the card holder, that is called a chargeback.

In an IT context, the term chargeback still has to do with money, and the idea of getting money back. However, in practice the intention ideally is not about disputing costs, but rather about measuring what the costs are of the IT resources that have been requested. Measurement of requested resources means that there is a potential to charge back the users or groups of users who requested these resources.

So, for example, an IT department for an organization may be allocated a budget to run a cluster, meant for the benefit of the organization. The IT department may make the cluster available to many other departments. These other departments request cluster resources. The IT department measures the requested resources and charges back the associated costs.

If a cluster is used by several different departments in the organization, then a simple way to pay for resource requests is to spread the entire cost as a general overhead expense over all departments equally. That may make matters easy for the cluster administrator, but can be harmful to the organization, because without a fair resource request management, there is a tendency for resource request abuse.

If however the costs per department are measured, it means that department managers can be kept aware of how resource requests are being divided up. Being able to measure requested resources per department, and thus being able to charge back to department for the requested resources, means that the organization using the cluster can plan and manage resource request budgets efficiently and fairly.

16.1.2 Comparison Of Job Chargeback Monitoring Measurement With Other Monitoring Measurements

Monitoring measurements in the cluster manager in general can be considered to be:

- Monitoring of devices (Chapter 13), which is about monitoring devices in a cluster.
- Job monitoring (Chapter 14), which is about using monitoring to measure the resources used by a jobs that actually run.
- Job accounting (Chapter 15), which is about using monitoring to measure the resources used by a user, a group, or other aggregation entity.
- Job chargeback (this chapter), which is about using monitoring to measure the resources requested for a job, whether the resources are used or not. The allows the requester to be charged the costs of requesting those resources.
The difference between actual resource use and requested resource use can be illustrated by a thought experiment:
First, a job is run that runs the CPU at 100% for 10 minutes. After that job is completed, a second job is run that runs a `sleep` command that lasts 10 minutes.
If these two jobs are considered from the point of view of CPU resource usage, then according to the workload manager:
The first job actually uses the CPU for 10 minutes, and the administrator can work out from the job monitoring system what user that ran that job, and charge the user for CPU usage.
For the second job, the administrator cannot use the job monitoring system to charge the user for the CPU usage because there was no significant usage measured. However, resources were used up during this time, because the request for the job blocked the availability of that CPU during this period for other users. The administrator would like to charge the user for preventing others from accessing the resources during that period. To do that requires measuring the resources requested during that period, rather than resources that were really used.
Thus, the aim of job chargeback monitoring is to provide a way to track resource requests, which typically differs a little from resource usage.

16.2 Job Chargeback Measurement
16.2.1 Predefined Job Chargebacks
Some predefined chargebacks can be listed and configured under the `chargeback` submode of the `wlm` mode.
The predefined list is short:

```
[bright92->wlm[slurm]]% chargeback
[bright92->wlm[slurm]->chargeback]% list
Name (key) Group by user Group by account Price per CPU second
------------------------------------------ ------------- ---------------- -------------------
Jobs completed last month grouped by user yes no 8.64$/d
Jobs completed this month grouped by user yes no 8.64$/d
Jobs completed this year grouped by user yes no 8.64$/d
```

Setting A Price For Resource Requests
The pricing can be set per resource requests over a specified time period. In the preceding list, the period is a month or a year. The resource and associated resource request consumption can be the following pairs:

<table>
<thead>
<tr>
<th>Resource And Resource Request Consumption Pairs</th>
</tr>
</thead>
<tbody>
<tr>
<td>Resource</td>
</tr>
<tr>
<td>---------</td>
</tr>
<tr>
<td>CPU</td>
</tr>
<tr>
<td>GPU</td>
</tr>
<tr>
<td>CPU core</td>
</tr>
<tr>
<td>slot</td>
</tr>
<tr>
<td>memory bytes</td>
</tr>
</tbody>
</table>

Different workload managers use different resources for resource measurement, which is why there is a variety in the resources that can be used for pricing.
16.2.2 Setting A Custom Job Chargeback

In addition to the predefined job chargebacks, more chargebacks can be added. For example, the number of jobs completed so far today can be set up as follows:

**Example**

```bash
[bright92->wlm[slurm]->chargeback]% add "Jobs completed this day grouped by user"
[bright92->...ck*[Jobs completed this day grouped by user*]]% show
```

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>Name</td>
<td>Jobs completed this day grouped by user</td>
</tr>
<tr>
<td>Revision</td>
<td>&lt;OB&gt;</td>
</tr>
<tr>
<td>Notes</td>
<td>no</td>
</tr>
<tr>
<td>Group by user</td>
<td>no</td>
</tr>
<tr>
<td>Group by group</td>
<td>no</td>
</tr>
<tr>
<td>Group by account</td>
<td>no</td>
</tr>
<tr>
<td>Group by job name</td>
<td>no</td>
</tr>
<tr>
<td>Group by job ID</td>
<td>no</td>
</tr>
<tr>
<td>Group by parent ID</td>
<td>no</td>
</tr>
</tbody>
</table>

**Users**

**Accounts**

**Job names**

**Job IDs**

**Parent IDs**

| Price per CPU second | 0.00$/s |
| Price per CPU core second | 0.00$/s |
| Price per GPU second | 0.00$/s |
| Price per memory byte-second | 0$/B*s |
| Price per slot second | 0$/slot*s |
| Currency | $ |

**Start time**

**End time**

**UTC**

**Include running**

**Calculate prediction**

```bash
[bright92->...ck*[Jobs completed this day grouped by user*]]% set pricepercpusecond 0.0001$/s
[bright92->...ck*[Jobs completed this day grouped by user*]]% set groupbyuser yes
[bright92->...ck*[Jobs completed this day grouped by user*]]% set starttime now/d
[bright92->...ck*[Jobs completed this day grouped by user*]]% set endtime now/d
[bright92->...ck*[Jobs completed this day grouped by user*]]% commit
```

**Chargeback Groupings**

The grouping for the new chargeback Jobs completed this day grouped by user is set to be Group by user. Grouping by user is a common grouping, because finding resource use by an individual user is typically the most useful case. Grouping is possible by:

- user
- group
- account
- job name
- job ID
• parent ID

After setting up chargebacks to suit the needs of the cluster administrator, queries can be made and reports can be generated using chargebacks. The `report` and `request` commands (section 16.2.3) are used for this.

### 16.2.3 The `report` And `request` Commands

Continuing on with the chargeback jobs completed this day grouped by user created in section 16.2.2, the `report` and `request` commands can be used after CPU request data values have been gathered on the jobs being run.

#### The `report` Command And Its Options

The `report` command displays a table of the number of jobs that were run per grouping for a chargeback, alongside the resource use and cost for each grouping.

Thus, some time after running jobs in Slurm, the `report` output for the chargeback created earlier, jobs completed this day grouped by user, might look as follows (some columns elided for clarity):

**Example**

```
[bright92->wlm[slurm]->chargeback[jobs completed this day grouped by user]]% report
# Start - Tue Sep 8 00:00:00 2020 CEST (1599516000)
# End - Tue Sep 8 23:59:59 2020 CEST (1599602399)

<table>
<thead>
<tr>
<th>User</th>
<th>Jobs</th>
<th>Runtime (s)</th>
<th>CPU (s)</th>
<th>CPU ($)</th>
<th>...</th>
<th>Price ($)</th>
</tr>
</thead>
<tbody>
<tr>
<td>alice</td>
<td>17</td>
<td>7,806</td>
<td>7,806</td>
<td>0.78</td>
<td>...</td>
<td>0.78</td>
</tr>
<tr>
<td>bob</td>
<td>20</td>
<td>8,775</td>
<td>8,775</td>
<td>0.88</td>
<td>...</td>
<td>0.88</td>
</tr>
<tr>
<td>charlie</td>
<td>19</td>
<td>7,007</td>
<td>7,007</td>
<td>0.7</td>
<td>...</td>
<td>0.7</td>
</tr>
<tr>
<td>david</td>
<td>10</td>
<td>5,122</td>
<td>5,122</td>
<td>0.51</td>
<td>...</td>
<td>0.51</td>
</tr>
<tr>
<td>edgar</td>
<td>25</td>
<td>10,502</td>
<td>10,502</td>
<td>1.05</td>
<td>...</td>
<td>1.05</td>
</tr>
<tr>
<td>frank</td>
<td>21</td>
<td>8,289</td>
<td>8,289</td>
<td>0.83</td>
<td>...</td>
<td>0.83</td>
</tr>
</tbody>
</table>
```

The help text for `report` command lists formatting options:

```
[bright92->wlm[slurm]->chargeback]% help report
Name: report - Create charge back report
Usage: report [options] <name>
Options:
- -d, --delimiter
  Set default row separator
- -v, --verbose
  Be more verbose: multiline table
- --start
  Pagination start offset
- --limit
  Pagination result limit
```

#### The `request` Command And Its Options

The `request` command lists the chargeback resources requested for a workload manager.

The `request` command can be run without options. In that case the output shows the resource request consumption for the chargeback, for the jobs over the period associated with that chargeback:
Example

[bright92->wlm[slurm]->chargeback[jobs completed this day grouped by user]]% request
# Start - Sun Sep 13 00:00:00 2020 CEST (1599948000)
# End - Sun Sep 13 23:59:59 2020 CEST (1600034399)

<table>
<thead>
<tr>
<th>Jobs</th>
<th>Runtime (s)</th>
<th>CPU (s)</th>
<th>Core (s)</th>
<th>GPU (s)</th>
<th>Slots (s)</th>
<th>Memory (B*s)</th>
</tr>
</thead>
<tbody>
<tr>
<td>5</td>
<td>1,765</td>
<td>1,765</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
</tbody>
</table>

The help text for request lists formatting, grouping, pricing, and filtering options:

[bright92->wlm[slurm]->chargeback]% help request

Name:
request - Chargeback report for workload

Usage:

request [options]

Options:

-u, --group-by-user
  group by username

--filter-user <user>[,<user>,...]
  filter on specified users

-g, --group-by-group
  group by group name

--filter-group <group>[,<group>,...]
  filter on specified groups

-a, --group-by-account
  group by account

--filter-account <account>[,<account>,...]
  filter on specified accounts

-j, --group-by-job-name
  group by job name

--filter-job-name <job-name>[,<job-name>,...]
  filter on specified job names

-i, --group-by-job-id
  group by job id

--filter-job-id <job-id>[,<job-id>,...]
  filter on specified job ids

-p, --group-by-parentid
  group by parent id

--filter-parent-id <parent-id>[,<parent-id>,...]
  filter on specified parent-ids

--price-per-cpu-second
Price per CPU second

--price-per-cpu-core-second
  Price per CPU core second

--price-per-gpu-second
  Price per GPU second

--price-per-gpu-second
  Price per GPU second

--price-per-memory-byte-second
  Price per memory byte * second

--price-per-slot-second
  Price per slot second

--currency
  Change the currency in which the price is displayed (default $)

--include-running
  include running jobs in charge back report (prices will not be final)

--calculate-prediction
  calculate a prediction for an incomplete time frame

-d, --delimiter
  Set default row separator

--sort <field1>[,<field2>,....]
  Override default sort order

--start-time, -s <time>
  Start time in Prometheus format

--end-time, -e <time>
  End time in Prometheus format, falls back to start-time if not specified

--utc
  Use UTC instead of local time

--start
  Pagination start offset

--limit
  Pagination result limit

-v, --verbose
  Be more verbose

Examples:
request Chargeback report for workload for the current WLM
request default Chargeback report for workload for default

For example, using epoch times to specify the start and end times, and grouping by user:
Example

```
[bright92->wlm[slurm]->chargeback[jobs completed this day grouped by user]]% request -s 1599948000 -e 1600034399 -u
# Start - Sun Sep 13 00:00:00 2020 CEST (1599948000)
# End - Sun Sep 13 23:59:59 2020 CEST (1600034399)
User Jobs Runtime (s) CPU (s) Core (s) GPU (s) Slots (s) Memory (B*s)
--------- ----- ------------ ------------ --------- ---------- ------------ ------------
alice 1 398 398 0 0 0 0
 david 2 1,041 1,041 0 0 0 0
edgar 1 325 325 0 0 0 0
frank 1 1 1 0 0 0 0
```

Another way of duplicating the output of `request` without options, is to explicitly specify the default values. For the chargeback jobs completed this day grouped by user, which was set up in section 16.2.2, it corresponds to a start time of `now/d` and an end time of `now/d`:

Example

```
[bright92->...eback[jobs completed this day grouped by user]]% request
# Start - Tue Sep 15 00:00:00 2020 CEST (1600120800)
# End - Tue Sep 15 23:59:59 2020 CEST (1600207199)
Jobs Runtime (s) CPU (s) Core (s) GPU (s) Slots (s) Memory (B*s)
------------ ------------ ------------ ------------ ------------ ------------ ------------
56 72,374 72,374 0 0 0 0
```

Users can be added to the table with the `-u` option (some output is truncated here for clarity in the examples that follow):

Example

```
[bright92->...eback[jobs completed this day grouped by user]]% request -u
# Start - Tue Sep 15 00:00:00 2020 CEST (1600120800)
# End - Tue Sep 15 23:59:59 2020 CEST (1600207199)
User Jobs Runtime (s) CPU (s) Core (s) GPU (s)
--------- ----- ------------ ------------ ----------
alice 16 19,538 19,538 0 0
bob 7 15,158 15,158 0 0
charlie 6 6,056 6,056 0 0
david 3 1,636 1,636 0 0
edgar 15 16,475 16,475 0 0
frank 9 13,511 13,511 0 0
```

A jobs drilldown can be carried out with `-j`.

Example

```
[bright92->wlm[slurm]->chargeback[jobs completed this day grouped by user]]% request -j
# Start - Tue Sep 15 00:00:00 2020 CEST (1600120800)
# End - Tue Sep 15 23:59:59 2020 CEST (1600207199)
Job name Jobs Runtime (s) CPU (s) Core (s) GPU (s)
```
The jobs drilldown can be carried out for a particular user, alice, using the filter option:

**Example**

```
[bright92->...grouped by user]]% request -j --filter-user alice
```

```
# Start - Tue Sep 15 00:00:00 2020 CEST (1600120800)
# End - Tue Sep 15 23:59:59 2020 CEST (1600207199)
Job name     Jobs Runtime (s) CPU (s) Core (s) GPU (s)
------------ ------------ ------------ ------------ ------------
data-transfer 5 13,404 13,404 0 0
iozone       6 3,132 3,132 0 0
sleep        5 3,002 3,002 0 0
```

This can have the user fields added for the case when several users are specified, as follows:

**Example**

```
[bright92->...grouped by user]]% request -s now/d -e now/d -u -j --filter-user alice,bob,charlie
```

```
# Start - Tue Sep 15 00:00:00 2020 CEST (1600120800)
# End - Tue Sep 15 23:59:59 2020 CEST (1600207199)
User    Job name     Jobs Runtime (s) CPU (s) Core (s) GPU (s)
------- -------------- ------------ ------------ ------------ ------------
alice   data-transfer 5 13,404 13,404 0 0
alice   iozone       6 3,132 3,132 0 0
alice   sleep        5 3,002 3,002 0 0
bob     data-transfer 3 12,842 12,842 0 0
bob     iozone       1 516 516 0 0
bob     sleep        3 1,800 1,800 0 0
charlie data-transfer 1 3,212 3,212 0 0
charlie iozone       2 1,044 1,044 0 0
charlie sleep       3 1,800 1,800 0 0
```

As the help text for `request` suggests, there are many more combinations possible.

### 16.3 Job Chargeback Background Information

Because users can run large numbers of jobs per day, the storage requirement for chargeback records can be very large indeed. For this reason, a cache is kept in memory, and flushed to storage in a MySQL database.

**CMDaemon AdvancedConfig** directives for configuring the MySQL storage for chargebacks are:

- `JobInformationChargeBackKeepDuration` (page 870)
- `JobInformationChargeBackKeepCount` (page 871)
- `JobInformationChargeBackRemoveInterval` (page 871)
Day-to-day Administration

This chapter discusses several tasks that may come up in day-to-day administration with the cluster manager.

Section 17.1 discusses running shell commands in parallel over the cluster.
Section 17.2 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 17.3 discusses how backups can be implemented for the cluster manager.
Section 17.4 discusses revision control for images.
Section 17.5 discusses BIOS configuration with the cluster manager.
Section 17.6 discusses checking hardware matching across the nodes of the cluster.
Section 17.7 discusses Serial Over LAN console access.
Section 17.8 discusses administrative aspects of handling the large amounts of raw monitoring data.
Section 17.9 discusses node replacement.
Section 17.10 discusses using Ansible to configure the cluster via Ansible collections and playbooks.

17.1 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 17.1.1), runs from within the OS shell. That is, pdsh executes its commands from within bash by default.

- **pexec** (parallel execute, section 17.1.2, 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.

### 17.1.1 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
-N name... select one or more misc modules to initialize first
-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@bright92 ~]# cmsh -c "device status"
node001 .................. [ UP ]
node002 .................. [ DOWN ]
bright92 ................. [ 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@bright92 ~]# pdsh -A hostname
node001: node001
node002: ssh: connect to host node002 port 22: No route to host
pdsh@bright92: node002: ssh exited with exit code 255
bright92: bright92

-v: With this option an attempt is made to run the command only on nodes nodes that are in the state UP:

Example
[root@bright92 ~]# pdsh -A -v hostname
node001: node001
bright92: bright92

-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@bright92 ~]# pdsh -v -g computenode hostname
node001: node001
This option allows a node list (\texttt{man pdsh(1)}) to be specified on the command line itself:

**Example**

```
[root@bright92 ~]# pdsh -w node00[1-2] hostname
node001: node001
node002: ssh: connect to host node002 port 22: No route to host
pdsh@bright92: node002: ssh exited with exit code 255
```

This option is the converse of \texttt{-w}, and excludes a node list that is specified on the command line itself:

**Example**

```
[root@bright92 ~]# pdsh -x node002 -w node00[1-2] hostname
node001: node001
```

**The \texttt{dshbak} Command**

The \texttt{dshbak} (distributed shell backend formatting filter) command is a filter that reformats \texttt{pdsh} output. It comes with the \texttt{pdsh} package.

Running \texttt{dshbak} with the \texttt{-h} option displays:

```
[root@bright92 ~]# 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 \texttt{man dshbak(1)}.

For the examples in this section, it is assumed that all the nodes in the cluster are now up. That is, \texttt{node002} used in the examples of the preceding section is now also up. Some examples to illustrate how \texttt{dshbak} works are then the following:

**Without \texttt{dshbak}**:

**Example**

```
[root@bright92 ~]# pdsh -A ls /etc/services /etc/yp.conf
bright92: /etc/services
bright92: /etc/yp.conf
node001: /etc/services
node001: ls: cannot access /etc/yp.conf: No such file or directory
pdsh@bright92: node001: ssh exited with exit code 2
node002: /etc/services
node002: /etc/yp.conf
```

**With \texttt{dshbak}, with no \texttt{dshbak} options**:

**Example**

```
[root@bright92 ~]# pdsh -A ls /etc/services /etc/yp.conf | dshbak
node001: ls: cannot access /etc/yp.conf: No such file or directory
pdsh@bright92: node001: ssh exited with exit code 2
----------------
bright92
----------------
```
17.1 Parallel Shells: pdsh And pexec

With dshbak, with the -c (coalesce) option:

Example

[root@bright92 ~]# pdsh -A ls /etc/services /etc/yp.conf | dshbak -c
node001: ls: cannot access /etc/yp.conf: No such file or directory
pdsh@bright92: node001: ssh exited with exit code 2
----------------
node002,bright92
----------------
/etc/services
/etc/yp.conf
[root@bright92 ~]#

The dshbak utility is useful for creating human-friendly output in clusters with larger numbers of nodes.

17.1.2 pexec In cmsh

In cmsh, the pexec command is run from device mode:

Example

[bright92->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

17.1.3 pexec In Bright View

In Bright 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 Bright 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 Bright View is faster in most cases, due to less graphics rendering overhead.

17.1.4 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

[bright92->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

[bright92->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

[bright92->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)
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.

### 17.1.5 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:

```
[bright92->device] help <pcommand>
```

### 17.2 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 technical help from Bright Computing or Bright Computing’s resellers, or a release update. This section describes how to get support for technical issues, and notifications for release updates.

#### 17.2.1 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 Bright 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 Bright 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:


- Cluster administrators that do not have a Bright 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 Bright Computing, it is true that many of the principles covered there are also useful when sending a query to Bright Computing when trying to resolve an issue. A read through the document may therefore be helpful before submitting a query to Bright Computing.

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 Bright Computing is to welcome such reports, to provide feedback to the reporter, and to
resolve the problem.

If there is apparently no response from Bright Computing 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 Bright Computing 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, Bright Computing also provides the cm-diagnose (section 17.2.2)
and the request-remote-assistance (section 17.2.3) utilities to help resolve issues. The use of third-
party screen-sharing utilities (section 17.2.4) is also possible.

17.2.2 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 Bright Computing 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@bright92 ~]# cm-diagnose
To be able to contact you about the issue, please provide
your e-mail address (or support ticket number, if applicable):
frankfurter@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 Bright Support asks for this.
Do you want to create a gdb trace of the running CMDaemon? [y/N]

Proceed to collect information? [Y/n]

Processing master
  Processing commands
  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/bright92_609.tar.gz

Cleaning up

Automatically submit diagnostics to
http://support.brightcomputing.com/cm-diagnose/ ? [Y/n] y

Uploaded file: bright92_609.tar.gz
Remove log file (/root/bright92_609.tar.gz)? [y/N] y
[root@bright92 ~]

17.2.3 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 Bright Computing 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 Bright Computing 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 Bright Computing’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@bright92 ~]# request-remote-assistance
This tool helps securely set up a temporary ssh tunnel to sandbox.brightcomputing.com.

Allow a Bright Computing engineer ssh access to the cluster? [Y/n]

Enter additional information for Bright Computing (eg: related ticket number, problem description)? [Y/n]

End input with ctrl-d
Ticket 1337 - the florbish is grommicking

Thank you.

Added temporary Bright 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 Bright Computing Support.
Closing this window will terminate the remote assistance session.

-----------------------------------------------
Hostname: bright92.NOFQDN
Connected on port: 7000

ctrl-c to terminate this session
17.2 Getting Support With Cluster Manager Issues, And Notifications For Release Updates

Bright Computing 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@bright92 ~]# 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 bright92
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.brightcomputing.com terminated.
Removed temporary Bright public key.
[root@bright92 ~]#
```

The Bright Computing 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 Bright Computing’s internet servers.

17.2.4 Requesting Remote Support With A Shared Screen Utility

If the request-remote-assistance utility (section 17.2.3) is restricted or otherwise disallowed by the site policy, then it may be permitted to allow a Bright support engineer to access the site using a third party shared screen utility that the administrator is comfortable with. In that case, possible options include:

- Zoom (https://zoom.us) (preferred)
- WebEx (https://www.webex.com)
- join.me (https://www.join.me)

Other screensharing options may also be possible. The support engineer and the administrator need to agree upon the option to be used, and decide upon a time (and time zone) for the shared screen session.

17.2.5 Getting Notified About Updates

Updates for the various NVIDIA Bright 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 17.1). 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.
17.3 Backups

17.3.1 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 18.4.8.

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**

```plaintext
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](http://www.rsnapshot.org).

  Rsnapshot requires access to port 22 on the head node.

### 17.3.2 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 And Repairs

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. Detected corruption causes an event notice to be sent to `cmsh` or Bright View.

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.

If a basic repair is to be carried out on a database, CMDaemon should first be stopped.

**Example**

```
[root@headnode ~]# service cmd stop
[root@headnode ~]# myisamchk --recover /var/lib/mysql/mysql.user.MYI
[root@headnode ~]# service cmd start
```

If basic repair fails, more extreme repair options—`man myisamchk(1)` suggests what—can then be tried out.

Another example: If CMDaemon is unable to start up due to a corrupted database, then messages in the `/var/log/cmdaemon` file might show something like:
Example

Oct 11 15:48:20 solaris CMDaemon: Info: Sending reconnect command to all\nodes which were up before master went down ...

Here it is CMDaemon’s “Database corruption” message that the administrator should be aware of, and which suggests database repairs are required for the CMDaemon database. The severity of the corruption, in this case not even allowing CMDaemon to start up, may mean a restoration from backup is needed. How to restore from backup is explained in the next section.

Restoring From The Local Backup
If the MySQL database repair tools of the previous section do not fix the problem, then, for a failover configuration, the dbreclone option (section 18.4.2) should normally provide a CMDaemon and Slurm database that is current. The dbreclone option does not clone the monitoring database.

Cloning the monitoring database: The cm-clone-monitoring-db.sh helper script that comes with CMDaemon can be used to clone the monitoring database.

Cloning extra databases: The file /cm/local/apps/cluster-tools/ha/conf/extradbcclone.xml template can be used as a template to create a file extradbcclone.xml in the same directory. The extradbcclone.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 17.8).

17.4 Revision Control For Images

NVIDIA Bright 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.

17.4.1 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 mod-
ified, 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 vari-
ations on the fly (section 3.16.4) 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 recom-
mended, 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.

17.4.2 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 Bright Computing 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 781.

17.4.3 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@bright92 ~]# 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@bright92 ~]# cd /cm/
[root@bright92 cm]# mv images images2
[root@bright92 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@bright92 cm]# mkfs.btrfs /dev/sdc1
[root@bright92 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@bright92 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@bright92 cm]# mkfs.btrfs butter.img
WARNING! - Btrfs Btrfs v0.20-rc1 IS EXPERIMENTAL
WARNING! - see http://btrfs.wiki.kernel.org before using
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` or `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@bright92 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@bright92 cm]# btrfs filesystem show /dev/loop0
Label: none uuid: 6f94de75-2e8d-45d2-886b-f87336b73474
  Total devices 1 FS bytes used 3.42GB
devid 1 size 20.00GB used 6.04GB path /dev/loop0
```

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 Bright View.
17.4.4 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 were 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 `<parent software image name>@<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>`
  Running `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 17.4.3. 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**

```
[bright92->softwareimage]% newrevision default-image "some changes"
[bright92->softwareimage]% newrevision default-image "more changes"
[bright92->softwareimage]% revisions default-image
```
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**
   
   `[bright92->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**
   
   `[bright92->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**
   
   `[bright92->category] % set storage softwareimage default-image@latest`
   
   then nodes in the `storage` category use the image `default-image@1` as their image 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 17.4.3.

First, a revision of the image is made to save its initial state:

```
[bright92->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:

```
[bright92->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:

```
[bright92->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:

```
[bright92->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:

```
[bright92->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.

17.5 BIOS Configuration And Updates

17.5.1 BIOS Configuration Via CMDaemon And Redfish

The cluster manager uses a CMDaemon API to manage the BIOS of nodes in userspace.

In NVIDIA Bright Cluster Manager version 9.0 and earlier, the implementation varied according to the proprietary vendor implementation.

Starting with NVIDIA Bright 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_r730.json
- dell_r740.json
- hpe_dl380.json

Example

[root@bright92 ~]# cd /cm/local/apps/cm-bios-tools/templates/
[root@bright92 templates]# ls -al
total 32
drwxr-xr-x 2 root root 50 Jul 21 20:59 .
-drwxr-xr-x 4 root root 34 Jul 21 20:59 ..
-rw-r--r-- 1 root root 5674 Jul 21 20:59 dell_r730.json
-rw-r--r-- 1 root root 5674 Jul 21 20:59 dell_r740.json
-rw-r--r-- 1 root root 23208 Jul 21 20:59 hpe_dl380.json
[root@bright92 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. Selecting 'UEFI' enables booting to Unified Extensible Firmware Interface (UEFI) capable operating systems. Selecting '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. Operating 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. Default: 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. Early Snoop is best used for latency sensitive workloads. This setting offers the best balance between workload effects. Home Snoop is best used for NUMA workloads that need maximum local and remote memory bandwidth. Cluster on Die is best used for highly NUMA optimized workloads. This setting offers the best case local memory latency, but worst case remote latency. Cluster On Die is only available when Node Interleaving is Disabled. Opportunistic Snoop Broadcast, available on select processor models, works well for workloads of mixed NUMA optimization. It offers a good balance of latency and bandwidth. Default: 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
Bright --------------> CMDaemon database --------------> BIOS (pending) --------------> BIOS (live)
View

For example, the cluster administrator might adjust the BIOS configuration for the node or category
17.5 BIOS Configuration And Updates

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 725). However the cmsh or Bright 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

[bright92->device[node002]->biossettings]% status
Parameter Configured Pending Live
-------------------------------- ---------- ------- ----
Pending errors:
No model defined

It can be set with the help of tab-completion:

Example

[bright92->device[node002]->biossettings]% set model<tab><tab>
dell_r730  hpe_dl380
[bright92->device[node002]->biossettings]% set model hpe_dl380
[bright92->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

[bright92->device[node002]]% biossettings
[bright92->device[node002]]% 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 > - CDW 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

[bright92->device[node002]->biossettings]% status |head -2 ; status |grep "Boot Mode"

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Configured</th>
<th>Pending</th>
<th>Live</th>
</tr>
</thead>
<tbody>
<tr>
<td>Boot Mode</td>
<td>&lt; default &gt;</td>
<td>-</td>
<td>UEFI Mode</td>
</tr>
</tbody>
</table>

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

[bright92->device[node002]->biossettings]% set boot mode <tab><tab>
legacy bios mode uefi mode
[bright92->device[node002]->biossettings]% set boot mode legacy bios mode
[bright92->device[*node002]->biossettings]% commit
[bright92->device[node002]->biossettings]% status |head -2 ; status |grep "Boot Mode"

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Configured</th>
<th>Pending</th>
<th>Live</th>
</tr>
</thead>
<tbody>
<tr>
<td>Boot Mode</td>
<td>Legacy BIOS Mode</td>
<td>-</td>
<td>UEFI Mode</td>
</tr>
</tbody>
</table>
• 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

```
[bright92->device[node002]->biossettings]% ..
[bright92->device[node002]]% bios check
```

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Configured</th>
<th>Detected</th>
</tr>
</thead>
<tbody>
<tr>
<td>different</td>
<td>BootMode</td>
<td>LegacyBios</td>
</tr>
</tbody>
</table>

The bios check command shows a result if there is a difference between the configured (CM-Daemon) and detected (live) configuration.

2. The next step is to apply the configuration change to the Pending BIOS state:

Example

```
[bright92->device[node002]]% bios apply

Node Result Output Error
------------ -------- -------------------------------- --------------------------------
node002 good
```

3. Finally, a reboot causes the Pending value to be made live:

Example

```
[bright92->device[node002]->biossettings]% ..
[bright92->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

```
[bright92->device[node002]->biossettings]% status |head -2 ; status |grep "Boot Mode"
```

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Configured</th>
<th>Pending</th>
<th>Live</th>
</tr>
</thead>
<tbody>
<tr>
<td>Boot Mode</td>
<td>&lt; default &gt;</td>
<td>-</td>
<td>Legacy BIOS Mode</td>
</tr>
</tbody>
</table>

**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 723) or Bright View front ends.

The cm-bios-manage help text is:
usage: cm-bios-manage [-h]
-a | -c | -f VENDOR_MODEL | -p VENDOR_MODEL | -P VENDOR_MODEL PROFILE | -t | -T VENDOR_MODEL] [-l] [-d]

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@bright92 ~]# /cm/local/apps/cm-bios-tools/python/cm-bios-manage -t
[
  "dell_r730",
  "hpe_d1380"
]
[root@bright92 ~]# /cm/local/apps/cm-bios-tools/python/cm-bios-manage -p hpe_d1380
[
  "test"
]
[root@bright92 ~]# /cm/local/apps/cm-bios-tools/python/cm-bios-manage -P hpe_d1380 test
{
  "test": "xyz"
}
[root@bright92 ~]# /cm/local/apps/cm-bios-tools/python/cm-bios-manage -T hpe_d1380
[
  {
    "displayName": "High Precision Event Timer (HPET) ACPI Support",
    "name": "AcpiHpet",
    "pos": {
      "w": 12,
      "r": 0,
      "o": 0,
      "g": 0
    }
  }, ...
]
17.5 BIOS Configuration And Updates

17.5.2 Updating BIOS Versions

There are two ways that the firmware can be updated. A legacy way based on DOS tools (page 727), and a more recent way, based on CMDaemon and Redfish (page 727).

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/
Day-to-day Administration

- **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**

```
[bright92->device]% firmware info

<table>
<thead>
<tr>
<th>Node</th>
<th>Filename</th>
<th>Component</th>
<th>Version</th>
<th>State</th>
<th>Progress</th>
<th>Result</th>
<th>Size</th>
<th>Date</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>iLO5-2.42</td>
<td>iLO5</td>
<td>2.42</td>
<td>undefined</td>
<td>N/A</td>
<td>8.6MiB</td>
<td></td>
<td>02/28/2022, 11:22:55</td>
</tr>
<tr>
<td></td>
<td>iLO5-2.43</td>
<td>iLO5</td>
<td>2.43</td>
<td>undefined</td>
<td>N/A</td>
<td>8.7MiB</td>
<td></td>
<td>02/28/2022, 11:22:55</td>
</tr>
<tr>
<td></td>
<td>iLO5-2.44</td>
<td>iLO5</td>
<td>2.44</td>
<td>undefined</td>
<td>N/A</td>
<td>8.8MiB</td>
<td></td>
<td>02/28/2022, 11:22:55</td>
</tr>
<tr>
<td></td>
<td>iLO5-2.45</td>
<td>iLO5</td>
<td>2.45</td>
<td>undefined</td>
<td>N/A</td>
<td>8.9MiB</td>
<td></td>
<td>02/28/2022, 11:22:55</td>
</tr>
</tbody>
</table>

[bright92->device]% firmware list -n node001

[bright92->device]% firmware upload iLO5-2.42 -n node001

<table>
<thead>
<tr>
<th>Node</th>
<th>Result</th>
<th>Output</th>
<th>Error</th>
</tr>
</thead>
<tbody>
<tr>
<td>node001</td>
<td>good</td>
<td>uploading: iLO5-2.42</td>
<td></td>
</tr>
</tbody>
</table>

[bright92->device]% firmware list -n node001

[bright92->device]% firmware flash iLO5-2.42 -n node001

<table>
<thead>
<tr>
<th>Node</th>
<th>Result</th>
<th>Output</th>
<th>Error</th>
</tr>
</thead>
<tbody>
<tr>
<td>node001</td>
<td>good</td>
<td>flashing: iLO5-2.42</td>
<td></td>
</tr>
</tbody>
</table>

[bright92->device]% firmware status -n node001

<table>
<thead>
<tr>
<th>Node</th>
<th>Filename</th>
<th>Component</th>
<th>Version</th>
<th>State</th>
<th>Progress</th>
<th>Result</th>
<th>Size</th>
<th>Date</th>
</tr>
</thead>
<tbody>
<tr>
<td>node001</td>
<td>iLO5</td>
<td>2.42</td>
<td>flashing</td>
<td>35.6%</td>
<td>N/A</td>
<td>02/28/2022, 11:27:27</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
```

```
17.6  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):

```bash
[root@bright92 ~]# cmsh -c "category add newbunch; commit"
[root@bright92 ~]# 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:

```bash
(export device; for i in {129..255}: do
    echo "set node00$i category newbunch"
done
export "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:

```bash
[root@bright92 ~]# /cm/local/apps/cmd/scripts/healthchecks/node-hardware-profile -n node129 -s newbunch
```

The profile is intended to be the reference hardware against which all the other nodes should match, and is saved under the directory `/cm/shared/apps/cmd/hardware-profiles/`, and further under the directory name specified by the `-s` option, which in this case is newbunch.

3. The hardware-profile data producer (section 13.2.10) can then be enabled, and the sampling frequency set as follows:

```bash
[root@bright92 ~]# cmsh
[bright92]% monitoring setup use hardware-profile
[bright92->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`:

```
[bright92->monitoring->setup[hardware-profile]]% nodeexecutionfilters
[bright92->...-profile]->nodeexecutionfilters]% add category filterhwp
[bright92->...-profile]->nodeexecutionfilters*[filterhwp*]% set categories newbunch
[bright92->...-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.

### 17.7 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 Bright 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 Bright View or `cmsh`.
   - In Bright 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 17.7.1)
   - the configuration of SOL with Bright View (section 17.7.2)
   - the configuration of SOL with `cmsh` (section 17.7.3)
   - the `conman` SOL logger and viewer (section 17.7.4)
17.7.1 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, CMC, 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 17.7.2 and 17.7.3 explain how SOL access configuration is set up for the node kernel using Bright 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 Bright 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 17.7.3).** 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.**

### 17.7.2 SOL Console Configuration With Bright View

In Bright 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**.

### 17.7.3 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@bright92 ~]# cmsh
[bright92]# softwareimage use default-image
```

```
[bright92->softwareimage*[default-image]]% set enablesol yes
[bright92->softwareimage*[default-image*]]% commit
```
The SOL settings for a particular image can be seen with the `show` command:

```
[bright92->softwareimage[default-image]]% show | grep SOL
```

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>Enable SOL</td>
<td>yes</td>
</tr>
<tr>
<td>SOL Flow Control</td>
<td>yes</td>
</tr>
<tr>
<td>SOL Port</td>
<td>ttyS1</td>
</tr>
<tr>
<td>SOL Speed</td>
<td>115200</td>
</tr>
</tbody>
</table>

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@bright92 ~]# cmsh
[bright92]% device use node001
[bright92->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:
```

### 17.7.4 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 Bright 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 17.2).

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 17.3) during the freeze of the physical console display.
Figure 17.3: Remote Console Continues During SOL Access During Physical Console Freeze

Finally, just before login is displayed, the physical console once more (figure 17.4) starts to display what is on the remote console (figure 17.5).
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>`.

### 17.8 Managing Raw Monitoring Data

From NVIDIA Bright Cluster Manager version 8.0 onwards, 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.

#### 17.8.1 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**

```
[bright92->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 0.00% -
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%.

#### 17.8.2 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**
### 17.8.3 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 863) 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`).

### 17.8.4 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 13.4.1) to smaller, but still non-zero values. After re-initializing the monitoring database, so that existing data is removed, the values reported by the `cm-bright-monitoring-usage.py` script (section 17.8.2) then show the new storage estimates for the monitoring data.

### 17.8.5 Deleting All Monitoring Data

A procedure to delete all monitoring data from the default `/var/spool/cmd/monitoring/` directory is as follows:

1. On the active head node, the CMDaemon service can be stopped with:

   ```bash
   pdsh -g all service cmd stop
   ```

2. On both head nodes, the monitoring data is removed with:

   ```bash
   rm -f /var/spool/cmd/monitoring/*
   ```
3. On both head nodes, the associated database tables for the CMDaemon user are cleared with a 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:

```bash
[root@bright92 ~]# 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:

```bash
[root@bright92 ~]# 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**

```bash
[root@bright92 ~]# 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]> exit
```

4. On both head nodes, CMDaemon can then be restarted with:

   ```bash
   service cmd start
   ```

5. On the active head node, after the command:

   ```bash
   cmha status
   ```

   shows all is OK, the CMDaemon service can be started on the active node again. The OK state should be achieved in about 15 seconds.

   The CMDaemon service is started with, for example:

   ```bash
   pdsh -g computenode service cmd start
   ```

### 17.9 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@bright92 ~]# cmsh
[root@bright92 ~]# device use node0031
if new mac address is known, then:
[root@bright92->device[node031] ~]# set mac <new mac address>
else if new mac address is not known:
[root@bright92->device[node031] ~]# clear mac
the changed setting in either case must be committed:
[root@bright92->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 \texttt{newnodes} command of \texttt{cmsh} (page 219) and the \texttt{Nodes Identification} resource of Bright View (page 223).

17.10 Ansible And NVIDIA Bright Cluster Manager

This section describes using Ansible with the cluster manager. Using Ansible to install NVIDIA Bright Cluster Manager is described in section 3.4 of the \textit{Installation Manual}.

17.10.1 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 \url{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 \textit{inventory} of machines.

- An Ansible \textit{module} is code, usually in Python, that is executed by Ansible to carry out Ansible \textit{tasks}, usually on a remote node. The module returns values.

- An Ansible \textit{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 \textit{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 \url{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 \url{galaxy.ansible.com}.

Picking Up The Bright Ansible Collections

In particular, the web interface at \url{https://galaxy.ansible.com/brightcomputing} shows the updated list of the cluster manager Ansible collections. They can be installed using the \texttt{ansible-galaxy} command line utility or downloaded directly as tarballs.
One minor issue with using the standard ansible-galaxy utility to pick up the Bright Ansible collection, is that the collection that is picked up by default may not be the latest. This is due to how the Galaxy repository parses the cluster manager packages, and presents the default package for downloading.

To make it easier to pick up a particular version ("semantic version") for an NVIDIA Bright Cluster Manager build, it can be picked up directly from the Bright Ansible collection server at http://ansible.brightcomputing.com/. For example, to pick up the latest collection for NVIDIA Bright Cluster Manager release 9.1.1, the collection can be picked up with:

Example

```
```

### 17.10.2 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 picked up:

```
[root@bright92 ~]# module load python3
[root@bright92 ~]# pip install ansible-base
```

The ansible Python interpreter is set:

```
[root@bright92 ~]# export ANSIBLE_PYTHON_INTERPRETER=/cm/local/apps/python3/bin/python
```

The connection is local in this case, so the `--connection=local` option is not needed for `ansible-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@bright92 ~]# cat /cm/local/examples/cmd/ansible/add-user.yaml
---
- hosts: all
  gather_facts: false

  tasks:
  - name: create test-user
    brightcomputing.bcm.user:
      name: test-user
      password: test-user-password
      profile: readonly
```

The appropriate version, for example 9.1.1, of the NVIDIA Bright Cluster Manager Ansible collection can be installed with the `ansible-galaxy` client from the ansible.brightcomputing.com repository directly:

Example

```
[root@bright92 ~]# ansible-galaxy collection install http://ansible.brightcomputing.com/brightcomputing-bcm-9.1.1.tar.gz
Starting galaxy collection install process
Process install dependency map
Starting collection install process
```
Installing 'brightcomputing.bcm:9.1.1+40698.gitcfbd1d8' to '/root/.ansible/collections/ansible_collections/brightcomputing/bcm'
brightcomputing.bcm (9.1.1+40698.gitcfbd1d8) was installed successfully

If the collection is instead picked up from the galaxy.ansible.com repository, then the version that is downloaded by default is unlikely to be the one that is wanted:

Example

[root@bright92 ~]# ansible-galaxy collection install brightcomputing.bcm
Starting galaxy collection install process
Process install dependency map
Starting collection install process
Installing 'brightcomputing.bcm:9.1.0-dev.1+40457.gite1c6b7c' to '/root/.ansible/collections/ansible_collections/brightcomputing/bcm'
Downloading https://galaxy.ansible.com/download/brightcomputing-bcm-9.1.0-dev.1+40457.gite1c6b7c.tar.gz to /root/.ansible/tmp/ansible-local-1405968e499qak/tmp1rlb753t
brightcomputing.bcm (9.1.0-dev.1+40457.gite1c6b7c) was installed successfully

Documentation For The Cluster Manager Collection

The brightcomputing.bcm documentation for modules can be explored using ansible-doc in the usual way, using the namespace. For example, for the user module in the brightcomputing.com namespace, this would be (output truncated):

[root@bright92 ~]# ansible-doc brightcomputing.bcm.user
  BRIGHTCOMPUTING.BCM.USER
  (/root/.ansible/collections/ansible_collections/brightcomputing/bcm/plugins/modules/user.py)

User

OPTIONS (= is mandatory):

- ID
  User ID number
  [Default: (null)]
  type: str

- certSerialNumber
  Serial number of certificate assigned to user
  [Default: -1]
  type: int

- 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

- commonName
  Full name (e.g. Donald Duck)
  [Default: (null)]
  type: str

- email

...
The list of modules in the `brightcomputing.bcm` collection can be viewed with `ansible-doc -l brightcomputing.bcm`.

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@bright92 ~]# ansible-playbook -i localhost. /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@bright92 ~]# cmsh -c "user list"
```

<table>
<thead>
<tr>
<th>Name</th>
<th>ID</th>
<th>Primary group</th>
<th>Secondary groups</th>
</tr>
</thead>
<tbody>
<tr>
<td>cmsupport</td>
<td>1000</td>
<td>cmsupport</td>
<td></td>
</tr>
<tr>
<td>test-user</td>
<td>1001</td>
<td>test-user</td>
<td></td>
</tr>
</tbody>
</table>

### 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 (17.10.3) and the next (17.10.4) 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 17.10.3) 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.
Carrying Out A YUM Update On The Default Software Image
The default software image can be updated with a playbook update.yaml defined and run as follows:

```
[root@bright92 ~]# cat update.yaml
- hosts: all
  gather_facts: false
  tasks:
    - name: Upgrade all packages
      dnf:
        name: "*"
        state: latest
        installroot: /cm/images/default-image

[root@bright92 ~]# unset ANSIBLE_PYTHON_INTERPRETER
[root@bright92 ~]# ansible-playbook -ilocalhost, update.yaml
PLAY [all]
... several lines of output skipped...
[root@bright92 ~]# export ANSIBLE_PYTHON_INTERPRETER=/cm/local/apps/python3/bin/python
```

Because the YUM update uses a built-in Ansible module, the interpreter is unset so that the distribution Python, provided by /usr/bin/python, is used, rather than the cluster manager Python. After running the playbook, the interpreter is switched back to the cluster manager Python.

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@bright92 ~]# cat /cm/local/examples/cmd/ansible/clone-software-image.yaml
---
- hosts: all
  gather_facts: false
  tasks:
    - name: clone a software image
      brightcomputing.bcm.software_image:
        name: cloned-image
        cloneFrom: default-image
        path: /cm/images/cloned-image

[root@bright92 ~]# ansible-playbook -i localhost, /cm/local/examples/cmd/ansible/clone-software-image.yaml
```

Cloning The Category
A clone of the default category, democategory, can be built with the brightcomputing.bcm.category module:

```
[root@bright92 ~]# cat clonedefaultcat.yaml
- hosts: all
  gather_facts: false
  tasks:
    - name: clone category
      brightcomputing.bcm.category:
```

name: democategory
cloneFrom: default

[root@bright92 ~] 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@bright92 ~] # cat setimageincat.yaml
- hosts: all
  gather_facts: false
  tasks:
    - name: set image in category
      brightcomputing.bcm.category:
        name: democategory
        softwareImageProxy:
          parentSoftwareImage: cloned-image

[root@bright92 ~] 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@bright92 ~] # cat setcatofnodes.yaml
- hosts: all
  gather_facts: false
  tasks:
    - name: list all nodes
      brightcomputing.bcm.node_info:
        format: dict
        include_id: false
        for_update: true
        register: result
    - name: set head node
      set_fact:
        all_nodes: "{{ result.nodes }}"
    - name: assign compute nodes to cloned category
      brightcomputing.bcm.physical_node:
        hostname: "{{ item }}"
        mac: "{{ all_nodes[item].mac }}"
        category: democategory
      loop:
        - node001
        - node002

[root@bright92 ~] ansible-playbook -i localhost, setcatofnodes.yaml

Without using Ansible, and using cmsh directly instead, the preceding placement could be carried out with:

[root@bright92 ~] for i in {001..002}
do cmsh -c "device use node$i; set category democategory; commit"
done
17.10.4 A More Complicated Playbook Example: Creating An Edge Site And Related Properties

This section provides a more complicated Bright 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 748.

The Collection

The full collection is as follows:

```yaml
- hosts: all
gather_facts: false
vars:
  site:
    name: test-site
    secret: SECRET

  director:
    hostname: test-site-director
    mac: 00:11:22:33:44:55
    eth0_ip: 10.152.0.254
    eth1_ip: 10.161.0.254

  nodes:
    - hostname: edge-node-01
      mac: 00:11:22:33:44:01
      eth0_ip: 10.161.0.1
    - hostname: edge-node-02
      mac: 00:11:22:33:44:02
      eth0_ip: 10.161.0.2
    - hostname: edge-node-03
      mac: 00:11:22:33:44:03
      eth0_ip: 10.161.0.3

pre_tasks:
  - name: set compute nodes for site
    set_fact:
      site_compute_nodes: "{{nodes | map(attribute='hostname') | list}}"
  - name: set nodes for site
    set_fact:
      site_nodes: "{{[director.hostname] + site_compute_nodes}}"

tasks:
  # Network creation
  - name: create an external network
    brightcomputing.bcm.network:
      state: present
      name: test-site-external_network
      type: EDGE_EXTERNAL
      baseAddress: 10.152.0.0
```
- name: create an internal network
  brightcomputing.bcm.network:
    state: present
    type: EDGE_INTERNAL
    name: test-site-internal_network
    baseAddress: 10.161.0.0
    broadcastAddress: 10.161.255.255
    dynamicRangeStart: 10.161.16.0
    dynamicRangeEnd: 10.161.19.255
    netmaskBits: 16
    domainName: test-site-internal_network
    management: true
    bootable: true

- name: create edge site software image
  brightcomputing.bcm.software_image:
    name: my-software-image
    path: /cm/images/my-software-image
    cloneFrom: default-image

- name: create edge director category
  brightcomputing.bcm.category:
    name: edge_director_category
    softwareImageProxy:
      parentSoftwareImage: my-software-image
      fsmounts:
        - device: $localnfsserver:/cm/shared
          mountpoint: /cm/shared
          filesystem: nfs
        - device: $localnfsserver:/home
          mountpoint: /home
          filesystem: nfs
    state: present

# Edge Director
- name: create director physical node
  brightcomputing.bcm.physical_node:
    state: present
    hostname: "{{director.hostname}}"
    partition: base
    interfaces_NetworkPhysicalInterface:
      - name: eth0
        ip: "{{ director.eth0_ip }}"
        network: test-site-external_network
      - name: eth1
        ip: "{{ director.eth1_ip }}"
        network: test-site-internal_network
    category: edge_director_category
    mac: "{{director.mac}}"
managementNetwork: test-site-external_network
provisioningInterface: eth0
installBootRecord: true
roles_EdgeDirectorRole:
  - name: edge_director_role
    openTCPPortsOnHeadNode: [636]
    externallyVisibleIp: 0.0.0.0
    externallyVisibleHeadNodeIp: 0.0.0.0
roles_BootRole:
  - name: boot_role
    allowRamdiskCreation: true
roles_StorageRole:
  - name: storage_role
roles_ProvisioningRole:
  - name: provisioning_role
    allImages: LOCALDISK

# Edge Nodes
- name: create edge nodes
  brightcomputing.bcm.physical_node:
    hostname: "{{item.hostname}}"
    softwareImageProxy:
      parentSoftwareImage: default-image
    interfaces_NetworkPhysicalInterface:
      - name: eth0
        ip: "{{item.eth0_ip}}"
        network: test-site-internal_network
    category: default
    mac: "{{item.mac}}"
  managementNetwork: test-site-internal_network
  installBootRecord: false
  provisioningInterface: eth0
  partition: base
  loop: "{{ nodes }}"

- name: add test edge site
  brightcomputing.bcm.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
    loop: "{{ 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:

```yaml
- 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.

```yaml
- name: create an external network #(1)
  brightcomputing.bcm.network:
    state: present
    name: test-site-external_network
    type: EDGE_EXTERNAL
    baseAddress: 10.152.0.0
    broadcastAddress: 10.152.255.255
    netmaskBits: 16
    domainName: test-site-external_network
    management: true

- name: create an internal network #(2)
  brightcomputing.bcm.network:
    state: present
    type: EDGE_INTERNAL
    name: test-site-internal_network
    baseAddress: 10.161.0.0
    broadcastAddress: 10.161.255.255
    dynamicRangeStart: 10.161.16.0
    dynamicRangeEnd: 10.161.19.255
    netmaskBits: 16
    domainName: test-site-internal_network
    management: true
    bootable: true
```

The possible values for each entity attribute can be seen by running the `ansible-doc` command:

Example

```
$ ansible-doc brightcomputing.bcm.network
```
750

Day-to-day Administration

Director creation: A category must exist for the edge director, or must be created before the director
can be created. The following snippet takes care of that:
- name: create edge site software image
brightcomputing.bcm..software_image:
name: my-software-image
path: /cm/images/my-software-image
cloneFrom: default-image
#(1)
- name: create edge director category
brightcomputing.bcm.category:
name: edge_director_category
softwareImageProxy:
parentSoftwareImage: my-software-image
fsmounts:
#(3)
- device: $localnfsserver:/cm/shared
mountpoint: /cm/shared
filesystem: nfs
- device: $localnfsserver:/home
mountpoint: /home
filesystem: nfs
state: present
#(4)

#(2)

1. The cloneFrom attribute is used to create the new software image from the existing one, to avoid
copying over all the values that are part of the original image. The default-image is used here,
since it is guaranteed to be defined in a new cluster.
The cloneFrom attribute only takes effect when the resource is not defined. This means that if the
software image is already present, then using cloneFrom has no effect. Removing the image allows
it to be re-created again using the cloneFrom attribute.
2. The declared software image (here it is my-software-image) is then used to define the director
category.
3. The edge directory category attributes in the snippet are standard values that are normally assigned to a director category that is to be used by director nodes.
4. The default value for state is present, so the task has the same behavior if the state field is left
out.
Values for the director node: The edge director node values can now be set
- name: create director physical node
brightcomputing.bcm.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}}"


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.

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.
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.
High Availability

18.0 Introduction

18.0.1 Why Have High Availability?
In a cluster with a single head node, the head node is a single point of failure for the entire cluster. It is often unacceptable that the failure of a single machine can disrupt the daily operations of a cluster.

High availability configuration for a head node is about configuring an extra head node to provide the head node services in a redundant manner. If one head node fails, then the other head node can take over, thus providing the same services with a minimum of downtime.

High availability can be set up for other types of nodes too.

18.0.2 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 18.5). 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 2.15 of the Cloudbursting Manual.
18.0.3 High Availability Usually Uses Shared Storage
HA is typically configured using shared storage (section 18.1.5), such as from an NFS service, which typically provides the /home directory on the active (section 18.1.1) head, and on the regular nodes.

18.0.4 Organization Of This Chapter
The remaining sections of this chapter are organized as follows:

- **HA On Head Nodes**
  - Section 18.1 describes the concepts behind HA, keeping the cluster manager configuration in mind.
  - Section 18.2 describes the normal user-interactive way in which the cluster manager implementation of a failover setup is configured.
  - Section 18.3 describes the implementation of the cluster manager failover setup in a less user-interactive way, which avoids using the Ncurses dialogs of section 18.2
  - Section 18.4 describes how HA is managed with the cluster manager after it has been set up.

- **HA On Regular Nodes**
  - Section 18.5 describes the concepts behind HA for regular nodes, and how to configure HA for them.

- **HA And Workload Manager Jobs**
  - Section 18.6 describes the support for workload manager job continuation during HA failover.

18.1 HA Concepts

18.1.1 Primary, Secondary, Active, Passive

**Naming:** In a cluster with an HA setup, one of the head nodes is named the *primary* head node and the other head node is named the *secondary* head node.

**Mode:** Under normal operation, one of the two head nodes is in *active* mode, whereas the other is in *passive* mode.

The difference between naming versus mode is illustrated by realizing that while a head node which is primary always remains primary, the mode that the node is in may change. Thus, the primary head node can be in passive mode when the secondary is in active mode. Similarly the primary head node may be in active mode while the secondary head node is in passive mode. As an aside: the definition for primary in HA for NVIDIA Bright Cluster Manager should not be confused with the definition for primary that is used by workload managers such as Slurm and PBS Pro when a failover mechanism is configured by the workload manager (section 7.2.4).

The difference between active and passive is that the active head takes the lead in cluster-related activity, while the passive follows it. Thus, for example, with MySQL transactions, CMDaemon carries them out with MySQL running on the active, while the passive trails the changes. This naturally means that the active corresponds to the master, and the passive to the slave, in the MySQL master-slave replication mode that MySQL is run as.

18.1.2 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.
18.1.3 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 Bright 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/UGE server (sgemaster)
18.1.4 Failover Network Topology

A two-head failover network layout is illustrated in figure 18.1.

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 18.1. 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.

**18.1.5 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 login 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 Bright View (section 18.4.6).

**18.1.6 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 18.1.7. 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 onwards. 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 18.2.4).

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.

18.1.7 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 18.4.6):

```
[root@bright92 ~]# cmsh
[bright92]% partition failover base
[bright92->partition[base]->failover]% set disableautomaticfailover yes
[bright92->partition[base]->failover]% commit
```

With Bright 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 18.4.2), then the failover sequence must also be initiated manually by the administrator.
18.1.8 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.

18.1.9 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, and then plumbing all that into all the hypervisors as a vnic (in the case of VMware).
- 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.

18.2 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 56 of the Installation Manual) on Bright View session logout. This means:

- If LDAP is managed by the cluster manager, then on logout, new user certificates are generated, and a new Bright 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 Bright 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 18.2.1): the configuration parameters are set for the shared interface and for the secondary head node that is about to be installed.
2. **Cloning** (section 18.2.2): the secondary head node is installed by cloning it from the primary head node.
3. **Shared Storage Setup** (section 18.2.3): the method for shared storage is chosen and set up.

An optional extra stage is:

4. **Automated Failover Setup** (section 18.2.4): Power control to allow automated failover is set up.

18.2.1 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 18.2.1-18.2.3, and possibly section 18.2.4) 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 18.2).
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 NVIDIA Bright Cluster Manager reseller, and activated (section 4.3 of the Installation Manual).
6. The virtual shared internal alias interface name and virtual shared internal IP alias address are set.

7. The virtual shared external alias interface name and virtual shared external IP alias address are set. For the external shared virtual IP address as well as for the external regular IP addresses, each head node external interface address must be a static IP addresses for the HA configuration. Attempting to use DHCP for external addresses in HA is not going to work.

8. The host name of the passive is set.

9. Failover network parameters are set. The failover network physical interface should exist, but the interface need not be up. The network name, its base address, its netmask, and domain name are set. This is the network used for optional heartbeat monitoring.

10. Failover network interfaces have their name and IP address set for the active and passive nodes.

11. The primary head node may have other network interfaces (e.g. InfiniBand interfaces, a BMC interface, alias interface on the BMC network). These interfaces are also created on the secondary head node, but the IP address of the interfaces still need to be configured. For each such interface, when prompted, a unique IP address for the secondary head node is configured.

12. The network interfaces of the secondary head node are reviewed and can be adjusted as required. DHCP assignments on external interfaces can be set by setting the value DHCP. If the primary head node has a DHCP-assigned IP address, then the input field for the secondary head node is set by default to the value DHCP.

13. A summary screen displays the planned failover configuration. If alterations need to be made, they can be done via the next step.

14. The administrator is prompted to set the planned failover configuration. If it is not set, the main menu of cmha-setup is re-displayed.

15. If the option to set the planned failover configuration is chosen, then a password for the MySQL root user is requested. The procedure continues further after the password is entered.

16. Setup progress for the planned configuration is displayed (figure 18.3).
18.2 HA Setup Procedure Using cmha-setup

17. Instructions on what to run on the secondary to clone it from the primary are displayed (figure 18.4).

18.2.2 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 18.4.8.

After the preparation has been done by configuring parameters as outlined in section 18.2.1, 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 18.4.8). 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 18.5).
4. The following command is executed (figure 18.6) on the node that is to become a failover clone:

```
/cm/cm-clone-install --failover
```

When doing a re-clone as in section 18.4.8, instead of a failover clone, then it is the `--clone` option that is used instead of the `--failover` option.

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 18.2.3) 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@bright92 ~]# cmha status
Node Status: running in active master mode

Failover status:
bright92* -> master2
  failoverping [ OK ]
  mysql  [ OK ]
  ping   [ OK ]
  status [ OK ]
master2 -> bright92*
  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.

18.2.3 Shared Storage Setup

After cloning the head node (section 18.2.2), the last basic stage of creating an HA setup is setting up shared storage. The available shared storage forms are NAS and DAS.
18.2 HA Setup Procedure Using cmha-setup

**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 18.1.5. 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 758).

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.
18.2.4 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:

```bash
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 18.4). Database administration with the dbreclone command (section 18.4) 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 Bright 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 18.4.2). 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.

18.3 Running cmha-setup Without Ncurses, Using An XML Specification

18.3.1 Why Run It Without Ncurses?
The text-based Ncurses GUI for cmha-setup is normally how administrators should set up a failover configuration.

The express mode of cmha-setup allows an administrator to skip the GUI. 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 GUI 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.

18.3.2 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>] <-l|-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.

- `-l|--initialize`: prepares a failover setup by setting values in the CMDaemon database to the values specified in the configuration file. This corresponds to section 18.2.1. 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 18.2.2.

- `-f|--finalize`: After the passive node is cloned as described in steps 1 to 10 of section 18.2.2, the finalize option is run on the active node to run the non-GUI finalize procedure. This is the non-GUI version of steps 11 to 13 of section 18.2.2.
  - `-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.
18.3.3 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@bright92 ~]# cd /cm/local/apps/cluster-tools/ha/conf
[root@bright92 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 18.2.1.

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 18.2.2.

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@bright92 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 ]
Checking for dedicated failover network ..... [ OK ]
```

A reboot has been issued on master2

The preceding corresponds to steps 11 to 13 of section 18.2.2.

Adding Storage:
Continuing with the session on the active, setting up a shared storage could be done with:

```
[root@bright92 conf]# cmha-setup -x -c failoverconf.xml -s nas
```

The preceding corresponds to carrying out the NAS procedure of section 18.2.3.

18.4 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.

18.4.1 Changing An Existing Failover Configuration

Changing an existing failover configuration is usually done most simply by running through the HA setup procedure of section 18.2 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 18.2.1.
18.4 Managing HA

18.4.2 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 Bright 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:

<table>
<thead>
<tr>
<th>HA Status</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>failoverping</td>
<td>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.</td>
</tr>
<tr>
<td>mysql</td>
<td>MySQL replication status</td>
</tr>
<tr>
<td>ping</td>
<td>the other head node is reachable over the primary management network. It uses ICMP ping.</td>
</tr>
<tr>
<td>status</td>
<td>CMDaemon running on the other head node responds to REST calls</td>
</tr>
</tbody>
</table>

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 18.2.4).

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 18.1.7), 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 236).

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.
18.4 Managing HA

**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 759) 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 Bright support.

**Example**

```
[root@bright92 ~]# cmha status
Node Status: running in active master mode

Failover status:
bright92* -> head2
  failoverping [ OK ]
  mysql    [ OK ]
  ping     [ OK ]
```
status    [ OK ]
head2 -> bright92*
  failoverping [ OK ]
  mysql    [ FAILED ] (11)
  ping     [ OK ]
  status   [ OK ]
[root@bright92 ~]# cmha dbreclone head2
Proceeding will cause the contents of the cmdaemon state database on head2 to be resynchronized from this node (i.e. bright92 -> head2)

Are you sure? [Y/N] Y
Waiting for CMDaemon (3113) to terminate...
[ OK ]
Waiting for CMDaemon to start...
[ 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 ]

Failover status:
bright92* -> head2
  failoverping [ OK ]
  mysql    [ OK ]
  ping     [ OK ]
  status   [ OK ]

head2 -> bright92*
  failoverping [ OK ]
  mysql    [ OK ]
  ping     [ OK ]
  status   [ OK ]

18.4.3 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
   ...

There are a number of possible states that a head node can be in:

<table>
<thead>
<tr>
<th>State</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>INIT</td>
<td>Head node is initializing</td>
</tr>
<tr>
<td>FENCING</td>
<td>Head node is trying to determine whether it should try to become active</td>
</tr>
<tr>
<td>ACTIVE</td>
<td>Head node is in active mode</td>
</tr>
<tr>
<td>PASSIVE</td>
<td>Head node is in passive mode</td>
</tr>
<tr>
<td>BECOMEACTIVE</td>
<td>Head node is in the process of becoming active</td>
</tr>
<tr>
<td>BECOMEPASSIVE</td>
<td>Head node is in the process of becoming passive</td>
</tr>
<tr>
<td>UNABLETOBECOMEACTIVE</td>
<td>Head node tried to become active but failed</td>
</tr>
<tr>
<td>ERROR</td>
<td>Head node is in error state due to unknown problem</td>
</tr>
</tbody>
</table>

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.

18.4.4 Failover Action Decisions
A table summarizing the scenarios that decide when a passive head should take over is helpful:
<table>
<thead>
<tr>
<th>Event on active</th>
<th>Reaction on passive</th>
<th>Reason</th>
</tr>
</thead>
<tbody>
<tr>
<td>Reboot</td>
<td>Nothing</td>
<td>Event is usually an administrator action. To make the passive turn active, an administrator would run <code>cmha makeactive</code> on it.</td>
</tr>
<tr>
<td>Shutdown</td>
<td>Nothing</td>
<td>As above.</td>
</tr>
<tr>
<td>Unusually sluggish or freezing system by state pingable with ICMP packets</td>
<td>Nothing</td>
<td>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 18.4.2).</td>
</tr>
<tr>
<td>Become passive in response to <code>cmha makeactive</code> run on passive</td>
<td>Become active when former active becomes passive</td>
<td>As ordered by administrator</td>
</tr>
<tr>
<td>Active dies</td>
<td>Quorum called, may lead to passive becoming new active</td>
<td>Confirms if active head is dead according to other nodes too. If so, then a “power off” command is sent to it. If the command is successful, the passive head becomes the new active head.</td>
</tr>
</tbody>
</table>

### 18.4.5 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 `provisioningnodeautoupdatetimeout` (page 203).

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 synchronization logic 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 `update provisioners` 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 Bright 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 12.2)
  - 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 18.2, 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 18.1.3 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 18.1.3.

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 18.4.2), 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.

18.4.6 High Availability Parameters

There are several HA-related parameters that can be tuned. Accessing these via Bright View is described in section 18.4.6. 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 18.1.7 covers this further.

Failover network
The Failover network setting determines which network is used as a dedicated network for the failover ping 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.
18.4.7 Viewing Failover Via Bright View

Accessing cmsh HA Parameters (partition failover base) Via Bright View

The Bright View equivalents of the cmsh HA parameters in section 18.4.6 are accessed from the clickpath Cluster → Partition[base] → Settings → Failover

18.4.8 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 18.2.2. Instead of running the cm-clone-install --failover command as in that section, the following command can be run:

```
[root@bright92 ~]# /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 773) before carrying out tasks with cmsh or Bright 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 18.2.2).

- **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**
18.5 HA For Regular Nodes And Edge Director Nodes

HA for regular nodes is available from NVIDIA Bright Cluster Manager version 7.0 onwards. HA for edge director nodes follows a similar design to HA for regular nodes, and is available from NVIDIA Bright Cluster Manager version 9.2 onwards.

18.5.1 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 manage 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 18.0.1).

18.5.2 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:

<table>
<thead>
<tr>
<th>HA For Head Nodes, Regular Nodes, And Edge Director Nodes: Some Features In Common</th>
</tr>
</thead>
<tbody>
<tr>
<td>Power control is needed for all HA nodes, in order to carry out automatic failover (section 18.1.7).</td>
</tr>
<tr>
<td>Warn time and dead time parameters can be set (section 18.4.7).</td>
</tr>
<tr>
<td>Mount and unmount scripts (page 758).</td>
</tr>
<tr>
<td>Pre- and post- failover scripts (section 18.4.6).</td>
</tr>
<tr>
<td>Disabling or enabling automatic failover (section 18.1.7).</td>
</tr>
<tr>
<td>A virtual shared IP address that is presented as the virtual node that is always up (section 18.1.4).</td>
</tr>
</tbody>
</table>
Some differences between head node HA and the other types of HA are:

<table>
<thead>
<tr>
<th>Head Node HA</th>
<th>Regular Node HA</th>
<th>Edge Director Node HA</th>
</tr>
</thead>
<tbody>
<tr>
<td>Installed with <code>cmha-setup</code> (section 18.2).</td>
<td>Installed by administrator using a procedure similar to section 18.5.3.</td>
<td>Installed by administrator using <code>cm-edge-setup</code> (section 2.1.1 of the Edge Manual)</td>
</tr>
<tr>
<td>Configurable within <code>failover</code> submode of <code>partition</code> mode (section 18.4.6).</td>
<td>Configurable within <code>failovergroups</code> submode of <code>partition</code> mode (section 18.5.3).</td>
<td>Configurable within <code>failovergroups</code> submode of <code>partition</code> mode (section 18.5.3).</td>
</tr>
<tr>
<td>Only one passive node.</td>
<td>Multiple passive nodes defined by failover groups.</td>
<td>Multiple passive nodes defined by failover groups.</td>
</tr>
<tr>
<td>Can use the optional failover network and <code>failoverping</code> heartbeat.</td>
<td>No failover network. Heartbeat checks done via regular node network.</td>
<td>No failover network. Heartbeat checks done via regular node network.</td>
</tr>
<tr>
<td>A quorum procedure (section 18.1.6). If more than half the nodes can only connect to the passive, then the passive powers off the active and becomes the new active.</td>
<td>Active head node does checks. If active regular node is apparently dead, it is powered off (STONITH). Another regular node is then made active.</td>
<td>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.</td>
</tr>
</tbody>
</table>

**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).

### 18.5.3 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**

```
[bright92->partition[base]->failovergroups]% status
No active failover groups
[bright92->partition[base]->failovergroups]% add cupsgroup
[bright92->partition*[base*]->failovergroups*[cupsgroup*]]% list
Name (key)   Nodes
------------------------ ------------------------
cupsgroup
```
By default, CUPS is provided in the standard image, in a stopped state. In Bright 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**

```
[bright92->...][cups group*] % set nodes node001..node002
[bright92->...][cups group*] % commit
[bright92->...][cups group*]
... Failover group cups group, make node001 become active
... Failover group cups group, failover complete. node001 became active
```

**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**

```
[bright92->partition[base]->failovergroups[cups group*]] % device
[bright92->device*] % foreach -n node001..node002 (services; add cups; \ 
set runif active; set autostart yes; set monitored yes)
[bright92->device*] % commit
Successfully committed 2 Devices
[bright92->device*]
```

**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**

```
[bright92->partition[base]->failovergroups] % status
Name        State  Active Nodes
------------ ------------ ------------
cups group   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**

```
[bright92->partition[base]->failovergroups] % use cups group
[bright92->...][failovergroups[cups group*]] % makeactive node002
node002 becoming active ...
[bright92->partition[base]->failovergroups[cups group*]] %
... Failover group cups group. make node002 become active
... node001: Service cups was stopped
... node002: Service cups was started
... Failover group cups group. failover complete. node002 became active
```
An alternative is to simply use the cmha utility (section 18.4.2):

Example

[root@bright92 ~]# 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

[bright92->partition[base]->failovergroups]% show cupsgroup

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>Automatic failover after graceful shutdown</td>
<td>no</td>
</tr>
<tr>
<td>Dead time</td>
<td>10</td>
</tr>
<tr>
<td>Disable automatic failover</td>
<td>no</td>
</tr>
<tr>
<td>Mount script</td>
<td>cupsgroup</td>
</tr>
<tr>
<td>Name</td>
<td>cupsgroup</td>
</tr>
<tr>
<td>Nodes</td>
<td>node001,node002</td>
</tr>
<tr>
<td>Postfailover script</td>
<td></td>
</tr>
<tr>
<td>Prefailover script</td>
<td></td>
</tr>
<tr>
<td>Revision</td>
<td></td>
</tr>
<tr>
<td>Unmount script</td>
<td></td>
</tr>
<tr>
<td>Warn time</td>
<td>5</td>
</tr>
</tbody>
</table>

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

[bright92->device]% foreach -n node001..node002 (interfaces; add alias \bootif:0; set ip 10.141.240.1; set startif active; set network internalnet)
[bright92->device]% commit
Successfully committed 2 Devices
[bright92->device]% foreach -n node001..node002 (interfaces; list)

<table>
<thead>
<tr>
<th>Type</th>
<th>Network device name</th>
<th>IP</th>
<th>Network</th>
</tr>
</thead>
<tbody>
<tr>
<td>alias</td>
<td>BOOTIF:0</td>
<td>10.141.240.1</td>
<td>internalnet</td>
</tr>
<tr>
<td>physical</td>
<td>BOOTIF [prov]</td>
<td>10.141.0.1</td>
<td>internalnet</td>
</tr>
</tbody>
</table>

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

[bright92->...interfaces*[BOOTIF:0*]]% set additionalhostnames cups
[bright92->...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:

```bash
# Only listen for connections from the local machine.
Listen localhost:631
...
# Show shared printers on the local network.
...
BrowseLocalProtocols
...
</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:

```bash
# Allow remote access
Port 631
...
# Enable printer sharing and shared printers.
...
BrowseAddress @LOCAL
BrowseLocalProtocols CUPS dnssec
...
</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 3.16.4
- by saving and using a new image with the modifications, as suggested for greater modifications in section 3.16.2, page 172.
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 18.2.4.

Example

```bash
ssh node001
echo c > /proc/sysrq-trigger
```

A passive node then takes over.

18.5.4 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 783):

<table>
<thead>
<tr>
<th>Sequence Of Events In Making Another HA Regular Node Active</th>
</tr>
</thead>
<tbody>
<tr>
<td>All</td>
</tr>
<tr>
<td>Active</td>
</tr>
<tr>
<td></td>
</tr>
<tr>
<td></td>
</tr>
<tr>
<td></td>
</tr>
<tr>
<td></td>
</tr>
<tr>
<td></td>
</tr>
<tr>
<td>Passive</td>
</tr>
<tr>
<td></td>
</tr>
<tr>
<td></td>
</tr>
<tr>
<td></td>
</tr>
<tr>
<td></td>
</tr>
<tr>
<td></td>
</tr>
<tr>
<td>All</td>
</tr>
</tbody>
</table>

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.

18.6 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>
<thead>
<tr>
<th>WLM</th>
<th>DAS</th>
<th>NAS</th>
</tr>
</thead>
<tbody>
<tr>
<td>Slurm</td>
<td>Y</td>
<td>Y</td>
</tr>
<tr>
<td>OGS</td>
<td>Y</td>
<td>Y</td>
</tr>
<tr>
<td>UGE</td>
<td>Y</td>
<td>Y</td>
</tr>
<tr>
<td>PBSPro</td>
<td>N</td>
<td>Y</td>
</tr>
<tr>
<td>LSF</td>
<td>N</td>
<td>Y</td>
</tr>
</tbody>
</table>

As the table shows, PBS Pro, 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.
19

The Jupyter Notebook Environment Integration

19.1 Introduction

This chapter covers the installation and usage of the Jupyter Notebook environment in the cluster manager.

An updated list of the supported Linux distributions and Jupyter functionalities can be found in the feature matrix at https://support.brightcomputing.com/feature-matrix/, under the Feature column, in the section for Jupyter features.

An overview of the concepts and terminology follows.

What Is Jupyter Notebook?

Jupyter Notebook (https://jupyter-notebook.readthedocs.io/), or Jupyter, is a client-server open-source application that provides a convenient way for a cluster user to write and execute notebook documents in an interactive environment.

In Jupyter, a notebook document, or notebook, is content that can be managed by the application. Notebooks are organized in units called cells and can contain both executable code, as well as items that are not meant for execution.

Items not meant for execution can be, for example: explanatory text, figures, formulas, or tables. Notebooks can also store the inputs and outputs of an interactive session.

Notebooks can thus serve as a complete record of a user session, interleaving code with rich representations of resulting objects.

These documents are encoded as JSON files and saved with the .ipynb extension. Since JSON is a plain text format, notebooks can be version-controlled, shared with other users and exported to other formats, such as HTML, LaTeX, PDF, and slide shows.

What Is A Notebook Kernel?

A notebook kernel (often shortened to kernel) is a computational engine that handles the various types of requests in a notebook (e.g. code execution, code completions, inspection) and provides replies to the user (https://jupyter.readthedocs.io/en/latest/projects/kernels.html). Usually kernels only allow execution of a single language. There are kernels available for many languages, of varying quality and features.

What Is JupyterHub?

Jupyter on its own provides a single user service. JupyterHub (https://jupyterhub.readthedocs.io/) allows Jupyter to provide a multi-user service, and is therefore commonly installed with it. JupyterHub is an open-source project that supports a number of authentication protocols, and can be configured in order to provide access to a subset of users.
What Is JupyterLab?

JupyterLab (https://jupyterlab.readthedocs.io/) is a modern and powerful interface for Jupyter. It enables users to work with notebooks and other applications, such as terminals or file browsers. It is open-source, flexible, integrated, and extensible.

JupyterLab works out of the box with JupyterHub. It can be used to arrange the user interface to support a wide range of workflows in data science, scientific computing, and machine learning.

JupyterLab is extensible with plugins that can customize or enhance any part of the interface. Plugins exist for themes, file editors, keyboard shortcuts, as well as for other components.

What Is A Jupyter Extension?

Several components of the Jupyter environment can be customized in different ways with extensions. Some types of extensions are:

  #ipython-extensions)

  config_overview.html#notebook-extensions)

  html)

Extensions are usually developed, bundled, released, installed, and enabled in different ways. Each extension provides a new functionality for a specific component. For example, JupyterLab extensions can customize or enhance any part of the JupyterLab user interface. Extensions can provide new themes, file viewers, editors and renderers for rich output in notebooks. They can also add settings, add keyboard shortcuts, or add items to the menu or command palette.

What Is Jupyter Enterprise Gateway?

By default, Jupyter runs kernels locally, which can exhaust server resources. A resource manager can be used to deal with this issue. Jupyter Enterprise Gateway (https://jupyter-enterprise-gateway.readthedocs.io/) (JEG) is a pluggable open-source framework that leverages local underlying resource managers, such as Slurm or Kubernetes, to distribute kernels across the compute cluster. In addition to scalability, JEG also provides an improved multi-user support, and more granular security for Jupyter. This makes Jupyter more useful for enterprise, scientific, and academic implementations.

In the cluster manager, all the technologies mentioned in these sections are combined to provide a powerful, customizable and user-friendly Jupyter Notebook web interface running on a lightweight, multi-tenant, multi-language, scalable and secure environment, ready for a wide range of enterprise scenarios.

For convenience, in the following sections, Jupyter is generally used to collectively refer to Jupyter Notebook, JupyterHub, JupyterLab and Jupyter Enterprise Gateway.

19.2 Jupyter Environment Installation

Bright distributes Jupyter via two packages: cm-jupyter and cm-jupyter-local.

cm-jupyter is installed in the /cm/shared directory, which is by default exported over NFS. As a result, Jupyter kernels can run on all the compute nodes, without a separate installation to those nodes. cm-jupyter provides Jupyter Notebook, JupyterHub, JupyterLab, Jupyter Enterprise Gateway and some extensions.

cm-jupyter-local provides the JupyterHub system service (cm-jupyterhub.service), and is therefore designed to be installed only on the node exposing users to the web login page for Jupyter. For
convenience, this node is called the login node. A login node is typically the head node, but any cluster node can be used.

Since compute nodes are not reachable via a web interface by default, it is the responsibility of the cluster administrator to configure access to these nodes if they are configured to be login nodes while Jupyter runs. That is, login nodes that are compute nodes must have their access configured by assigning IP addresses, configuring the firewall, opening Jupyter ports, and so on. However, if the Jupyter login node is the head node, then the cluster manager takes care of configuring the firewall to open the required ports and of ensuring that the resulting environment is working out of the box.

**Bright Jupyter Extensions**

For a default deployment of Jupyter, Bright installs and enables the following extensions to the Jupyter environment:

- **Jupyter Addons**: A Jupyter Notebook server extension that performs API calls to CMDaemon and manages other server extensions;
- **Jupyter Kernel Creator (section 19.5)**: A Jupyter Notebook server extension that provides a new interactive and user-friendly way to create kernels;
- **Jupyter VNC (section 19.8)**: A Jupyter Notebook server extension that enables remote desktops with VNC from notebooks;
- **JupyterLab Tools**: A JupyterLab extension that exposes Bright server extensions functionalities to the users and shows the Cluster View section;
- **Jupyter WLM Magic (section 19.9)**: An IPython extension that simplifies scheduling of workload manager jobs from the notebook.

### 19.2.1 Jupyter Setup

The `cm-jupyter-setup` script can be run on the head node of the cluster to deploy a working Jupyter environment with minimal effort. The script comes with the cluster manager’s `cm-setup` package. It has no prerequisites, and can be run before or after configuring any resource manager, such as Kubernetes or Slurm.

The `cm-jupyter-setup` script installs the two packages `cm-jupyter` and `cm-jupyter-local`. By default, the Jupyter environment initially contains only Jupyter’s default Python 3 kernel, which runs on the login node.

During setup, an administrator can deploy the Jupyter login interface on multiple nodes to evenly distribute the load across them. In this case the administrator must configure a load balancer to route users’ requests across those nodes.

These login nodes become members of the same configuration overlay, and therefore share the same Jupyter configuration, such as port numbers, authenticator, and so on.

By default, the Jupyter configuration file points to local SSL certificates. This means that if there are multiple Jupyter login interfaces, then each node uses its own SSL certificate.

### 19.2.2 Jupyter Architecture

The default Jupyter architecture deployed by `cm-jupyter-setup` is shown in figure 19.1.
In this architecture, the cm-jupyterhub.service provided by cm-jupyter-local starts JupyterHub on the port specified by c.JupyterHub.hub_port (default: 8901) on a login node (typically the head node).

JupyterHub then automatically spawns a dynamic proxy to route HTTP requests via Bright’s cm-npm-configurable-http-proxy package.

The proxy is the only process that listens for clients’ requests on Jupyter’s public interface, as specified by c.JupyterHub.port (default: 8000).

JupyterHub instructs the proxy on how requests should be dispatched by using its REST API. This is exposed at the port specified by c.ConfigurableHTTPProxy.port (default: 8902), typically defined within c.ConfigurableHTTPProxy.api_url.

At this stage, users accessing Jupyter with their browsers are either redirected to the shared Hub (e.g. for authentication), or to their dedicated single-user servers to run notebooks.


The cm-jupyter-setup script automatically installs JupyterLab and sets /lab as the default URL (c.Spawner.default_url) to redirect users to the new interface.

Finally, Jupyter Enterprise Gateway is integrated to spawn kernels on different nodes by configuring jupyterhub.spawner.LocalProcessSpawner as the default spawning mechanism (c.JupyterHub.spawner_class), and jupyterhub-singleuser-gw as the default spawning command (c.Spawner.cmd). When a new notebook is started, JEG scans for an available port, and spawns a kernel chosen by the user on an appropriate cluster node. This process is then connected to JupyterLab.

JupyterHub and its HTTP proxy are run as two root processes, while JupyterLab and Jupyter Enterprise Gateway are run as user processes.

The privileges of the kernels spawned by Jupyter Enterprise Gateway can be configured by cluster administrators, and depend on the underlying computational engine (e.g. Kubernetes). By default, all
the kernels configurable by Bright are run as user processes, and no privilege escalation is possible.

Communication between users’ browsers, JupyterHub, its HTTP proxy, and JupyterLab, is secured by default by JupyterHub. On the other hand, communication between JupyterLab, Jupyter Enterprise Gateway, and kernels, is secured by Bright.

Administrators can customize their Jupyter integration by setting some of the aforementioned options when the cm-jupyter-setup script runs. New values are automatically handled by the cluster manager and written to Jupyter configuration files.

Other configuration options can be found in /cm/local/apps/jupyter/current/conf/jupyterhub_config.py.

19.2.3 Verifying Jupyter Installation

The cm-jupyter-setup script automatically starts the cm-jupyterhub service.

Any user (not necessarily root) can then verify the installation is working as expected. Here an ordinary user, jupyterhubuser runs the checks.

It can take some time until the service is fully up and running, even if systemctl status cm-jupyterhub -l shows that the service is active:

Example

[jupyterhubuser@bright92 ~]$ systemctl status cm-jupyterhub -l

```
Name: cm-jupyterhub.service
Description: JupyterHub
Loaded: loaded (/usr/lib/systemd/system/cm-jupyterhub.service; static; vendor preset: disabled)
Active: active (running) since Sat 2021-12-11 01:55:23 CET; 2min 41s ago
Main PID: 1766342 (run.sh)
Tasks: 13 (limit: 23356)
Memory: 104.5M
```

A check can then be done to see that the Jupyter extensions provided by Bright are installed and enabled:

Example

[jupyterhubuser@bright92 ~]$ module load jupyter
Loading jupyter/13.0.0
Loading requirement: python39
[root@bright92 ~]# jupyter serverextension list
config dir: /cm/shared/apps/jupyter/13.0.0/etc/jupyter
   jupyterlab enabled
     - Validating...
     jupyterlab 3.1.18 OK
   cm_jupyter_kernel_creator enabled
     - Validating...
     cm_jupyter_kernel_creator 1.1.dev148+gaae7880 OK
   cm_jupyter_addons enabled
     - Validating...
     cm_jupyter_addons 1.1.dev20+g09bdeb9 OK
   cm_jupyter_vnc enabled
     - Validating...
     cm_jupyter_vnc 1.1.dev52+g3ccf8d5 OK
Once functioning, the Jupyter web interface is accessible with a browser using the HTTPS protocol on the specified port (figure 19.2):

![Figure 19.2: JupyterHub login screen](image)

Any user registered in the Linux-PAM system can log in to Jupyter. If needed, a test user `jupyterhubuser` with password `jupyterhubuser` can be created with, for example:

**Example**

```
[root@bright92 ~]# cmsh -c "user; add jupyterhubuser; set password jupyterhubuser; commit"
```

After the first login, a new single-user server is spawned (figure 19.3):

![Figure 19.3: JupyterHub starting single-user server](image)

Users are redirected to the JupyterLab interface and have access to Jupyter’s default Python 3 kernel (figure 19.4):
If using Kubernetes under Jupyter, then a user registered under the Linux-PAM system must be added separately via Kubernetes with `cm-kubernetes-setup` (section 19.5.1).

### 19.3 Jupyter Notebook Examples

The `cm-jupyter` package (section 19.2.1) provides a number of machine learning notebook examples that can be executed with Jupyter.

The notebooks include some applications developed with TensorFlow, PyTorch, MXNet, and other frameworks. The applications can be found in the `/cm/shared/examples/jupyter/notebooks/` directory:

```
[jupyterhubuser@bright92 ~]$ ls /cm/shared/examples/jupyter/notebooks/
Keras+TensorFlow2-addition.ipynb  R-iris.ipynb  TensorFlow-minigo.ipynb
MXNet-superresolution.ipynb     Spark-pipeline.ipynb
Pytorch-cartpole.ipynb          Spark+XGBoost-mortgage.ipynb
```

The datasets needed to execute these notebooks can be found in the `/cm/shared/examples/jupyter/datasets/` directory:

```
[jupyterhubuser@bright92 ~]$ ls /cm/shared/examples/jupyter/datasets/
880f8b8a6fd-mortgage-small.tar.gz  kaggle-iris.csv
```
Users can copy these examples to their home directories, create or choose appropriate kernels to execute them, and interactively run them from Jupyter. In order to edit notebooks, the write permissions must be kept during the copy.

The distributed examples typically only require the packages provided by Bright with the Data Science Add-on, such as TensorFlow, PyTorch, and MXNet.

It is the responsibility of users to make sure that the required modules are loaded by their Jupyter kernels. The list of frameworks and libraries required to run an example is usually available at the beginning of each notebook.

19.4 Jupyter Kernels

In Jupyter, kernels are defined as JSON files.

Any user that the cluster administrator has registered in the Linux-PAM system can list the available Jupyter kernels via the command line. The following example is run in the initial Jupyter environment:

Example

```
[jupyterhubuser@bright92 ~]$ module load jupyter
Loading jupyter/13.0.0
Loading requirement: python37
[jupyterhubuser@bright92 ~]$ jupyter kernelspec list
Available kernels:
  python3 /cm/shared/apps/jupyter/13.0.0/share/jupyter/kernels/python3
```

Each kernel directory contains a `kernel.json` file describing how Jupyter spawns that kernel:

Example

```
[jupyterhubuser@bright92 ~]$ ls /cm/shared/apps/jupyter/13.0.0/share/jupyter/kernels/*
/cm/shared/apps/jupyter/13.0.0/share/jupyter/kernels/python3/kernel.json
```

In addition to specifications for shared kernels, each user can define new personal ones in the home directory. By default, the Jupyter data directory for a user is located at `$HOME/.local/share/jupyter`.

This path can be verified with Jupyter by using the `--paths` option:

Example

```
[jupyterhubuser@bright92 ~]$ jupyter --paths
config:
  /home/jupyterhubuser/.jupyter
  /cm/local/apps/python37/etc/jupyter
  /cm/shared/apps/jupyter/13.0.0/etc/jupyter
  /etc/jupyter
data:
  /home/jupyterhubuser/.local/share/jupyter
  /cm/local/apps/python37/share/jupyter
  /cm/shared/apps/jupyter/13.0.0/share/jupyter
  /usr/share/jupyter
runtime:
  /home/jupyterhubuser/.local/share/jupyter/runtime
```

The simplest definition for a Python3 kernel designed to run on the login node is:

```json
{
  "argv": ["python", 
              "-m", 
              "ipykernel_launcher"]
}```
In the preceding kernel definition:

- **argv**: is the command to be executed to locally spawn the kernel
- **"display_name"**: is the name to be displayed in the JupyterLab interface
- **"language"**: is the supported programming language
- **"{connection_file}"** (https://jupyter-client.readthedocs.io/en/stable/kernels.html#connection-files) is a placeholder, and is replaced by Jupyter with the actual path to the connection file before starting the kernel.

The following kernel is Jupyter’s default Python 3 kernel distributed by Bright in the initial environment:

**Example**

```bash
[jupyterhubuser@bright92 ~]$ cat /cm/shared/apps/jupyter/current/share/jupyter/kernels/python3/kernel.json
{
  "argv": [
    "/cm/local/apps/python37/bin/python3.7",
    "-m",
    "ipykernel_launcher",
    "--InteractiveShellApp.extra_extension=cm_jupyter_wlm_magic",
    "--TerminalIPythonApp.extra_extension=cm_jupyter_wlm_magic",
    "-f",
    "{connection_file}"],
  "display_name": "Python 3",
  "language": "python",
  "env": {
    "PYTHONPATH": "/cm/shared/apps/jupyter/current/lib64/python3.7/site-packages:/cm/shared/apps/jupyter/current/lib/python3.7/site-packages"
  }
}
```

The two kernels are not very different. They differ from each other in the Python 3 binary path, the IPython extension (Jupyter WLM Magic), and the exported PYTHONPATH environment variable.

### 19.4.1 Jupyter Enterprise Gateway Kernels

Jupyter is designed to run both the kernel processes, as well as the user interface (JupyterLab or Jupyter Notebook) on the same host. The kernel `{connection_file}` is therefore stored in the `~/.local/share/jupyter/runtime` directory, or in the `/run` directory.

JupyterLab can delegate the task of spawning kernels to another component. In the cluster manager installation it is Jupyter Enterprise Gateway (JEG) that takes on this role, performing a complete lifecycle management of the kernel. That is: start, status monitoring, and termination.

JEG requires an extended `kernel.json` definition to describe a particular process-proxy module to handle the kernel.

A simple definition for a Python3 kernel designed to be scheduled via JEG is:
In this example, the "metadata" entry has been added. It includes "class_name", which defines a path to a Python class compatible with the JEG API; and "config", an initial arguments dictionary passed to the JEG process.

JEG can use several pluggable modules that are included in the Bright environment to interact with a wide range of resource managers, such as Kubernetes or Slurm. By doing so, kernels can be scheduled across compute nodes.

Bright recommends that JEG kernels are created and used with the Jupyter Kernel Creator (section 19.5) extension.

19.5 Jupyter Kernel Creator Extension

Creating or editing kernels can be cumbersome and error-prone for users, depending on the features of the execution context desired for their notebooks.

To provide a more user-friendly experience, Bright includes the Jupyter Kernel Creator extension in JupyterLab. This extension is accessed from the navigation pane in the JupyterLab interface, by clicking on the Bright icon. Jupyter Kernel Creator allows users to create kernels using the JupyterLab interface, without the need to directly edit JSON files. With this interface users can create kernels by customizing an available template according to their needs.

A template can be considered to be the skeleton of a kernel, with several preconfigured options, and other options that are yet to be specified. Common customizations for templates include environment modules to be loaded, workload manager queues to be used, number and type of GPUs to acquire, and so on.

Templates are usually defined by administrators according to cluster capabilities, programming languages and user requirements. Each template can provide different options for customizations.

Administrators often create different templates to take advantage of different workload managers, programming languages and hardware resources. For example, an administrator may define a template for scheduling Python kernels via Kubernetes, another one for R kernels via Slurm, and yet another one for Bash kernels via Platform LSF.

To simplify Jupyter configuration for administrators, Bright distributes a number of pre-defined templates with Jupyter Kernel Creator. These templates can be used for default configurations of Bright workload managers, and can be customized and extended for more advanced use.

Kernel templates defined by Bright can be found in the Jupyter Kernel Creator installation directory:

```
[jupyterhubuser@bright92 ~]$ ls /cm/shared/apps/jupyter/current/lib/python3.7/site-packages/cm_jupyter_kernel_creator/kerneltemplates/
```
Users can view available templates in the Jupyter web browser interface, within the \textit{KERNEL TEMPLATES} section of the dedicated Bright extensions panel (figure 19.5):

![JupyterLab Bright extensions section with kernel templates](image)

Figure 19.5: JupyterLab Bright extensions section with kernel templates

Templates provided by Bright are not listed in the panel, if no associated workload manager instance or if no Kubernetes instance has been deployed on the cluster.

Users can instantiate a template to create an actual kernel from the dedicated Bright extensions section (+ button).

A dialog is dynamically generated for the template being instantiated, and users are asked to fill a number of customization options defined by administrators (figure 19.6):
Figure 19.6: Jupyter kernel template customization screen

Once the template is completely customized, the kernel can be created. It automatically appears in the JupyterLab Launcher screen (figure 19.7) and can be used to run notebooks.
A user who lists available Jupyter kernels via the command line now sees the newly-created kernel:

```bash
[jupyterhubuser@bright92 ~]$ module load jupyter
Loading jupyter/13.0.0
  Loading requirement: python37
[jupyterhubuser@bright92 ~]$ jupyter kernelspec list
Available kernels:
  jupyter-eg-kernel-k8s-py-1f9pqoi4q4 /home/jupyterhubuser/.local/share/jupyter/kernels/jupyter-eg-kernel-k8s-py-1f9pqoiq4
  python3 /cm/shared/apps/jupyter/13.0.0/share/jupyter/kernels/python3
```

The new kernel directory will contain the JSON definition generated by the Jupyter Kernel Creator:

```bash
[jupyterhubuser@bright92 ~]$ cat .local/share/jupyter/kernels/jupyter-eg-kernel-k8s-py-1f8verf93/kernel.json
{
  "language": "python",
  "display_name": "Python on Kubernetes 210624185005",
  "metadata": {
    "process_proxy": {
```

---

**Figure 19.7:** JupyterLab Launcher screen with new kernel

---
Jupyter kernel names need not be unique. Users should therefore choose meaningful and distinguishable display names for their kernels. Doing so makes the JupyterLab Launcher screen easier to use.

For convenience, a summary of the available kernel templates and their requirements is shown in Table 19.1:

Table 19.1: Available Jupyter kernel templates for Bright and their requirements

<table>
<thead>
<tr>
<th>Template name</th>
<th>Requirement</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>jupyter-eg-kernel-k8s-bash</td>
<td>Kubernetes1</td>
<td>Bash via Kubernetes</td>
</tr>
<tr>
<td>jupyter-eg-kernel-k8s-py</td>
<td>Kubernetes1</td>
<td>Python 3.9 via Kubernetes</td>
</tr>
<tr>
<td>jupyter-eg-kernel-k8s-py-spark</td>
<td>Kubernetes2</td>
<td>Python 3.9 + Spark 3 via Kubernetes</td>
</tr>
<tr>
<td>jupyter-eg-kernel-k8s-r</td>
<td>Kubernetes3</td>
<td>R 3.4 via Kubernetes</td>
</tr>
<tr>
<td>jupyter-eg-kernel-k8s-cmjkop-ngc-py</td>
<td>Kubernetes</td>
<td>NGC images via Jupyter Kernel Operator</td>
</tr>
<tr>
<td>jupyter-eg-kernel-k8s-cmjkop-julia</td>
<td>Kubernetes</td>
<td>Jupyter official image via Jupyter Kernel Operator (using Julia)</td>
</tr>
<tr>
<td>jupyter-eg-kernel-k8s-cmjkop-py</td>
<td>Kubernetes</td>
<td>Jupyter official image via Jupyter Kernel Operator (Python)</td>
</tr>
</tbody>
</table>

...continues
19.5 Jupyter Kernel Creator Extension

### Table 19.1: Available Jupyter kernel templates...continued

<table>
<thead>
<tr>
<th>Template name</th>
<th>Requirement</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>jupyter-eg-kernel-k8s-cmjkop-py-spark</td>
<td>Kubernetes</td>
<td>Python + Spark via Jupyter Kernel Operator (using Python and Spark)</td>
</tr>
<tr>
<td>jupyter-eg-kernel-1sf-bash</td>
<td>Platform LSF</td>
<td>Bash via Platform LSF</td>
</tr>
<tr>
<td>jupyter-eg-kernel-1sf-py39</td>
<td>Platform LSF</td>
<td>Python 3.9 via Platform LSF</td>
</tr>
<tr>
<td>jupyter-eg-kernel-opempbs-bash</td>
<td>Open PBS</td>
<td>Bash via Open PBS</td>
</tr>
<tr>
<td>jupyter-eg-kernel-opempbs-py39</td>
<td>Open PBS</td>
<td>Python 3.9 via Open PBS</td>
</tr>
<tr>
<td>jupyter-eg-kernel-pbspro-bash</td>
<td>PBS Pro</td>
<td>Bash via PBS Pro</td>
</tr>
<tr>
<td>jupyter-eg-kernel-pbspro-py39</td>
<td>PBS Pro</td>
<td>Python 3.9 via PBS Pro</td>
</tr>
<tr>
<td>jupyter-eg-kernel-slurm-bash</td>
<td>Slurm</td>
<td>Bash via Slurm</td>
</tr>
<tr>
<td>jupyter-eg-kernel-slurm-py39</td>
<td>Slurm</td>
<td>Python 3.9 via Slurm</td>
</tr>
<tr>
<td>jupyter-eg-kernel-uge-bash</td>
<td>UGE</td>
<td>Bash via UGE</td>
</tr>
<tr>
<td>jupyter-eg-kernel-uge-py39</td>
<td>UGE</td>
<td>Python 3.9 via UGE</td>
</tr>
</tbody>
</table>

1. Docker image: `brightcomputing/jupyter-kernel-sample:k8s-py36-1.1.1`
2. Docker image: `brightcomputing/jupyter-kernel-sample:k8s-spark-py37-1.2.1`
3. Docker image: `brightcomputing/jupyter-kernel-sample:k8s-r-1.0.0`

DockerHub kernels page: [https://hub.docker.com/r/brightcomputing/jupyter-kernel-sample/tags](https://hub.docker.com/r/brightcomputing/jupyter-kernel-sample/tags)

19.5.1 Running Jupyter Kernels With Kubernetes

The Jupyter Kernel Operator (section 19.7) is the recommended way to run kernels in Kubernetes in the cluster manager. It allows users to run unmodified images; it takes care of communication with Jupyter Enterprise Gateway; and it manages the Jupyter kernel life-cycle, including cleaning up dead or orphaned kernels.

After installation and configuration by `cm-kubernetes-setup`, Jupyter Kernel Operator kernels (jupyter-eg-kernel-k8s-cmjkop-* ) appear in the JupyterLab interface in the template section.

Kernels which do not support Jupyter Kernel Operator are hidden.

The following Bright templates allow users to create and run Jupyter Enterprise Gateway kernels on compute nodes via Kubernetes:

### Table 19.2: Bright templates for creating and running Jupyter Enterprise Gateway kernels on cluster nodes via Kubernetes

<table>
<thead>
<tr>
<th>Template</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>jupyter-eg-kernel-k8s-cmjkop-ngc-py</td>
<td>NGC images via Jupyter Kernel Operator</td>
</tr>
<tr>
<td>jupyter-eg-kernel-k8s-cmjkop-py</td>
<td>Jupyter official image via Jupyter Kernel Operator (Python)</td>
</tr>
<tr>
<td>jupyter-eg-kernel-k8s-cmjkop-julia</td>
<td>Jupyter official image via Jupyter Kernel Operator (Julia)</td>
</tr>
</tbody>
</table>

...continues
By default, templates that do not support Jupyter Kernel Operator appear only if Kubernetes Jupyter Kernel Operator is not installed.

The administrator has to make sure that Kubernetes is correctly configured on the cluster, and that Pod Security Policy (PSP) is enabled (section 9.3.10).

Details on Kubernetes installation are provided in section 9.3.

The default configuration proposed by cm-kubernetes-setup is usually sufficient to run Kubernetes kernels created from Bright’s templates. However, it is the responsibility of the administrator to add users registered in the Linux-PAM system to Kubernetes.

For example, a test user jupyterhubuser can be added to Kubernetes with:

**Example**

```
[root@bright92 ~]# cm-kubernetes-setup --add-user jupyterhubuser
```

For every new user added, cm-kubernetes-setup automatically generates a dedicated namespace with a name in the form <user>-restricted. For instance, the command in the example above, creates the namespace: jupyterhubuser-restricted.

Cluster administrators are strongly recommended to review pods security policies for these dedicated namespaces.

In order to speed up kernel creation for the first user logging into JupyterLab, Bright recommends that all the relevant Kubernetes images are pre-loaded on the compute nodes that Jupyter Enterprise Gateway can contact.

Jupyter notebooks using Kubernetes kernels cannot natively use machine learning libraries and frameworks provided by Bright, even if installed on the software image used by Kubernetes. The machine learning libraries and frameworks have to be included in the chosen Kubernetes image.

### 19.5.2 Running Jupyter Kernels With Workload Managers

Bright’s Jupyter Enterprise Gateway kernels can be created and run by users on compute nodes via workload managers (WLMs). For convenience, Slurm is used as an example in this section. However, the same instructions are valid for the other WLMs listed in table 19.1.

The templates used to create Slurm kernels are Bright’s

- jupyter-eg-kernel-slurm-bash
- jupyter-eg-kernel-slurm-py39

which offer a Bash and a Python 3.9 environment respectively.

The administrator has to make sure that Slurm is correctly configured on the cluster. Slurm installation is described in section 7.3.

The default configuration that cm-wlm-setup suggests in Express mode is usually sufficient to run Slurm kernels created from Bright’s templates. If Slurm is configured to automatically detect GPUs, they will be listed as available resources while instantiating Slurm templates.

The Jupyter login node must be authorized to submit Slurm jobs. This is typically the case, since the JupyterHub login node is by default the head node, and cm-wlm-setup by default assigns the slurmsubmit role to the head node.
Finally, the administrator must make sure that relevant dependency packages are installed on the software image used by Slurm clients. The image used is typically for the compute nodes. Missing packages may cause kernels to fail at startup or at run time.

In particular, administrators need to install cm-python39 to use kernels based on the jupyter-eg-kernel-slurm-py39 template.

Jupyter notebooks using Slurm kernels can natively use the machine learning libraries and frameworks provided by Bright on Slurm clients, if these are available. Further details on machine learning packages installation on both the head node and the compute nodes are given in section 1.4 of the Machine Learning Manual.

19.6 Jupyter Kernel Creator Extension Customization

The Jupyter kernel templates described in section 19.5 are stored under the directory /cm/shared/apps/jupyter/current/lib/python3.9/site-packages/cm_jupyter_kernel_creator/kerneltemplates/.

The template of a particular Jupyter kernel in Bright’s Jupyter Kernel Creator extension is a directory containing at least two files: meta.yaml and kernel.json.j2. The kernel template can also require other files placed in the same directory, such as icons. These files are copied to the target user kernels directory upon kernel creation, after the template is instantiated.

The meta.yaml file includes all the parameters that can be substituted into kernel.json.j2 and is defined with the YAML format (https://yaml.org/).

The kernel.json.j2 file is a skeleton of the kernel.json file to be generated. It can contain some placeholders, and is defined with the Jinja2 format (https://jinja.palletsprojects.com/+).

19.6.1 Kernel Template Parameters Definition

The kernel template meta.yaml file defines all the parameters that can be used in kernel.json.j2.

It should contain three entries:

- **display_name**: the name that will be displayed for the kernel in the JupyterLab Launcher;
- **features**: a list of features that must be available on the cluster to show this kernel template to JupyterLab users in the Bright extensions panel;
- **parameters**: the variables for kernel.json.j2.

The display_name entry is an arbitrary string:

```yaml
display_name: "A simple kernel"
```

The features entry is a list of strings. The only possible values for these strings are:

- "kubernetes"
- "lsf"
- "pbspro"
- "slurm"
- "uge"

For example, a kernel template that only requires Kubernetes to work contains this line in its meta.yaml:

```yaml
features: ["kubernetes"]
```
The same principle applies to the other workload managers. If multiple features are listed, then the kernel template is displayed if at least one of them is available.

Finally, the parameters entry contains a dictionary of kernel parameters. The keys of the dictionary are parameter names that are used in `kernel.json.j2`. The values of the dictionary are options that help users choose a correct value for the parameter. Parameter options are also dictionaries.

A kernel template `meta.yaml` is thus structured as:

```
[...]
parameters:
  <parameter_name1>:
    <option_key>: <option_value>
    ...
    <option_key>: <option_value>
  <parameter_name2>:
    <option_key>: <option_value>
    ...
    <option_key>: <option_value>
  ...
  <parameter_nameN>:
    <option_key>: <option_value>
    ...
    <option_key>: <option_value>
[...]
```

Parameter names are arbitrary strings. Option keys are strings. The only possible values for these strings are:

- `type`
- `definition`
- `limits`

The option keys `type` and `definition` are mandatory. The option key `limits` is optional. For example, a kernel template with two parameters `foo` and `bar` looks as follows:

```
[...]
parameters:
  foo:
    type: <option_value>
    definition: <option_value>
    limits: <option_value>
  bar:
    type: <option_value>
    definition: <option_value>
    limits: <option_value>
[...]
```

The **type** Option

The `type` option key defines the type of the kernel parameter. The `type` option key only accepts one of the following string values:

- `num`: for numeric values (both float and integer are supported);
- `str`: for arbitrary strings;
The definition Option

The definition option key defines how the parameter value is retrieved and displayed.

The definition option key accepts dictionary-like values. Allowed string keys for the dictionary are:

- **display_name**: the name that is displayed for the parameter in the kernel customization dialog. It accepts an arbitrary string as a value;
- **getter**: how parameter values are retrieved. It only accepts as value one of the following strings:
  - **static**: the values for the parameter are pre-defined;
  - **shell**: the values for the parameter are the output of the shell script;
  - **python**: the values for the parameter are the output of a Python script;
- **values**: possible values for the parameter that are displayed in the kernel customization dialog. It accepts a list of arbitrary values;
- **default**: default value for the parameter. It accepts a value from values;
- **exec**: the script to be executed to fill values when getter is shell or python. It accepts a shell/Python script.

If type option value is list, then every line of getter with a setting of shell or python is treated as an element of the list.

The limits Option

The limits option key offers a way to apply bounds on values provided by users. This usually reduces the chances of making mistakes and helps users defining correct kernels before actually running them.

The limits option key accepts dictionary-like values according to the value chosen for the type option key.

If num is the type, then limits can contain:

- **min**: the minimum numeric value;
- **max**: the maximum numeric value.

If str or list is the type, then limits can contain:

- **min_len**: the minimum length of the string or the list;
- **max_len**: the maximum length of the string or the list.

For example, if every node in the cluster has no more than 4 GPUs, then the upper limit on the requested GPU number can be set to 4:

```python
[...]
parameters:
  gpus:
    type: num
    definition: <option_value>
    limits:
      min: 1
      max: 4
[...]```
These limits are not a security measure. They should be considered as convenient sanity checks for values entered while instantiating a kernel template. This is because users are always able to later directly edit the generated `kernel.json`, thereby ignoring such limits.

### 19.6.2 Kernel Template Parameters Usage

In the `kernel.json.j2` file of a kernel template, two types of variables can be defined: Jinja2 variables ([https://jinja.palletsprojects.com/en/master/templates/#variables](https://jinja.palletsprojects.com/en/master/templates/#variables)) and Python formatted string literals (also known as f-strings) ([https://docs.python.org/3/reference/lexical_analysis.html#f-strings](https://docs.python.org/3/reference/lexical_analysis.html#f-strings)).

They should not be mixed as they work on different steps of the template lifecycle:

- `{{ .. }}` and `{% .. %}`: These formats are used to specify Jinja2 expressions and statements. Jinja2 expressions and statements are substituted or computed once, when the user creates a kernel with Bright’s Jupyter Kernel Creator extension. The actual values will be those specified in the kernel customization dialog;

- `{ .. }`: This format is used to specify Python f-strings. F-strings are substituted every time a user starts the kernel for a Jupyter notebook. The actual values will be a path to the kernel connection file, a job ID in a WLM, a UID of the running kernel process, a unique name for a Kubernetes pod, etc.

Some process-proxies in Jupyter Enterprise Gateway may require another type of variables template. For example, the Kubernetes process-proxy requires multiple objects to be set up in its API. Several unique configuration files need to be created at runtime. Templates for such objects are located in the `templates/` directory, along with the `meta.yaml` and `kernel.json.j2` files. These templates are rendered every time a Kubernetes kernel is started for a Jupyter notebook. Variables for these templates are located inside the `config` section of the `kernel.json` file.

### 19.6.3 Kernel Template Creation Example

To illustrate the kernel variables syntax described in sections 19.6.1 and 19.6.2, a kernel template is now created from Jupyter’s default Python 3 kernel.

The new kernel template has to be placed in a suitable directory for Bright’s Jupyter Kernel Creator extension. This directory is the `kerneltemplates/` directory in the extension’s Python module:

```
[root@bright92 ~]# ls /cm/shared/apps/jupyter/current/lib/python3.9/site-packages/\ cm_jupyter_kernel_creator/kerneltemplates/
```

```
jupyter-eg-kernel-k8s-bash jupyter-eg-kernel-lsf-bash jupyter-eg-kernel-slurm-bash
jupyter-eg-kernel-k8s-py jupyter-eg-kernel-ksf-py39 jupyter-eg-kernel-slurm-py39
jupyter-eg-kernel-k8s-py-spark jupyter-eg-kernel-pbspro-bash jupyter-eg-kernel-uge-bash
jupyter-eg-kernel-k8s-r jupyter-eg-kernel-pbspro-py39 jupyter-eg-kernel-uge-py39
```

The content of the original Jupyter kernel directory, including its images, is copied. The `kernel.json` file is then renamed to `kernel.json.j2` to make it a Jinja2 template:

```
[root@bright92 ~]# cd /cm/shared/apps/jupyter/current/lib/python3.9/site-packages/\ cm_jupyter_kernel_creator/kerneltemplates/

[root@bright92 kerneltemplates]# cp -pr /cm/shared/apps/jupyter/current/share/jupyter/kernels/\ python3 python3-test

[root@bright92 kerneltemplates]# mv python3-test/kernel.json{,.j2}
```

The next step is to create the `meta.yaml` file with the kernel template parameters. In this example a request is made to have Slurm deployed on the cluster:

```
[root@bright92 kerneltemplates]# cat << EOF >python3-test/meta.yaml
---
```
The new kernel template is now available in JupyterLab (figure 19.8):

![JupyterLab interface with new kernel template](image)

Figure 19.8: New kernel template in the JupyterLab interface

The next step is to define kernel template parameters. The first one being added is `kernel_name`, which is the name of the kernel in the JupyterLab Launcher. The `meta.yaml` file is updated:

```yaml
display_name: "Python 3 demo kernel"
features: ["slurm"]
parameters:
  kernel_name:
    type: str
    definition:
      getter: static
      default: "Demo kernel"
      display_name: "Display name of the kernel"
```

The `display_name` definition in the `kernel.json.j2` is now changed to a Jinja2 placeholder matching this `meta.yaml` parameter with the appropriate Jinja2 syntax:

```json
{
  "display_name": "{{ kernel_name }}",
  ...
}
```

The JupyterLab interface changes and now prompts the user to enter a value, while providing a default option (figure 19.9):

![JupyterLab interface with new kernel template dialog](image)

Figure 19.9: New kernel template dialog during template instantiation

If the user clicks on the Create button, then a new directory is created in `~/.local/share/jupyter/kernels`. The content of the kernel template directory, except for the `meta.yaml` and `kernel.json.j2` files, is copied into this new directory. The kernel name provided by the user is substituted into the `kernel.json.j2` template, and the resulting `kernel.json` file is written into the new user kernel directory.

As a result, the new kernel then appears in the JupyterLab Launcher.
19.6.4 Extending Kubernetes Kernel Templates

Bright’s JEG kernel templates for Kubernetes can be extended with additional parameters.

In the following example, a Kubernetes kernel template is extended, to show a dropdown menu with additional environment variables in the kernel customization dialog. The selected variables are exported to the context of the running kernel and made available in the Jupyter notebook.

In order to preserve the original kernel template, the existing jupyter-eg-kernel-k8s-py template is copied to a user directory. The new customized kernel template directory could be later copied back to the original Bright Jupyter Kernel Creator extension templates directory, to make it available for all the users.

```
[jupyterhubuser@bright92 ]$ mkdir -p ~/.local/share/jupyter/kerneltemplates
[jupyterhubuser@bright92 ]$ cd ~/.local/share/jupyter/kerneltemplates
[jupyterhubuser@bright92 kerneltemplates]$ cp -pr /cm/shared/apps/jupyter/current/lib/python3.9/site-packages/cm_jupyter_kernel_creator/kerneltemplates/jupyter-eg-kernel-k8s-py/.jupyter-eg-kernel-k8s-py-demo
```

To distinguish the new kernel template from the original one, it is convenient to change the display name:

```
---
display_name: "Python on Kubernetes Demo"
features: ["kubernetes"]
[...]
```

The next step is to add a new parameter in the meta.yaml file:

```
[...]
parameters:
[...]
additional_vars:
[...]
type: list
definition:
  getter: shell
default: []
display_name: "Additional variables to export in the pod"
exec:
  - echo -e "VAR1=value1\nCUR_DATE=$(date +"%Y-%m-%d")"
[...]
```

Now, if the kernel template is instantiated, then a new parameter is displayed in the kernel customization dialog (figure 19.10):

![Figure 19.10: New environment variables dropdown menu](image)

The kernel customization dialog also shows that every output line of the script was rendered as an item of the menu.

To pass the selected environment variables to the actual container in the running pod, the kernel.json.j2 and templates/pod.yaml.j2 files must both be modified. This is due to the two-step template rendering mechanism:
1. The first rendering takes place when the user instantiates the kernel template. The kernel.json.j2 file becomes kernel.json, and kernel parameters are replaced with the values specified in the dialog.

2. The second rendering takes place when the user starts the kernel for a Jupyter notebook. The pod.yaml.j2 file and other Jinja2 files in the templates/ directory are rendered to the actual YAML representations of the Kubernetes objects.

In the example that follows, the kernel.json.j2 file is extended with the "additional_vars" kernel parameter:

```json
{
  ...
  "metadata": {
    "process_proxy": {
      ...
      "config": {
        ...
        "additional_vars": {{ additional_vars | tojson }},
      ...
      }
    }
  },
  ...
}
```

The last step requires the use of the Jinja2 loop syntax in the templates/pod.yaml.j2 file:

```yaml
[...]
spec:
  ...
  containers:
    - name: notebook
      ...
      env:
        ...
        {% for env_var in additional_vars %}
          {% set var, value = env_var.split("=") %}
          - name: {{ var }}
            value: "{% value %}"
        {% endfor %}
  ...
```

Now, if a kernel is instantiated with both the environment variables (figure 19.11) and then started from a notebook, then the variables are available in the process (figure 19.12):

![Figure 19.11: Environment variables are chosen in the kernel customization dialog](image-url)
19.6.5 Extending Workload Managers Kernel Templates

Bright’s JEG kernel templates for workload managers can be extended with additional parameters. For convenience, Slurm is used as example in this section. However, the same instructions are valid for the other WLMs listed in table 19.1.

In the following example, a Slurm kernel template is extended, to show a dropdown menu with additional environment variables in the kernel customization dialog. The selected variables are exported to the context of the running kernel and made available in the Jupyter notebook.

In order to preserve the original kernel template, the existing `jupyter-eg-kernel-slurm-py39` template is copied to a user directory. The new customized kernel template directory could be later copied back to the original Bright Jupyter Kernel Creator extension templates directory, to make it available for all users.

```
[jupyterhubuser@bright92 ~]# mkdir -p ~/.local/share/jupyter/kerneltemplates
[jupyterhubuser@bright92 ~]# cd ~/.local/share/jupyter/kerneltemplates
[jupyterhubuser@bright92 kerneltemplates]# cp -pr /cm/shared/apps/jupyter/current/lib/python3.9/site-packages/cm_jupyter_kernel_creator/kerneltemplates/jupyter-eg-kernel-slurm-py39 ./jupyter-eg-kernel-slurm-py39-demo
```

To distinguish the new kernel template from the original one, it is convenient to change the display name:

```
---
display_name: "Python 3.9 via SLURM Demo"
features: ["slurm"]
[...]
```

The next step is to add a new parameter in the `meta.yaml` file:

```
[...]
parameters:
[...]
additional_vars:
  type: list
  definition:
    getter: shell
    default: []
    display_name: "Additional variables to export in the job"
    exec:
      - echo -e "VAR1=value1\nCUR_DATE=$(date +"%Y-%m-%d")"
[...]
```

Now, if the kernel template is instantiated, then a new parameter is displayed in the kernel customization dialog (figure 19.13):
The kernel customization dialog also shows that every output line of the script was rendered as an item of the menu.

To pass the selected environment variables to the actual workload manager job, the "submit_script" section inside the kernel.json.j2 file must be modified to take into account the new "additional_vars" kernel parameter:

```json
{
    [...]
    "metadata": {
        "process_proxy": {
            [...]
            "config": {
                [...]
                "submit_script": [
                    [...]
                    "set -xe",
                    "{environ}",
                    {% for env in additional_vars %}
                    "export {{ env }}",
                    {% endfor %}
                    "{kernel_cmd}"
                ]
            }
        }
    },
    [...]
}
```

Now, if a kernel is instantiated with both the environment variables (figure 19.14) and then started from a notebook, then the variables are available in the process (figure 19.15):

```
Additional variables to export in the job:

VAR1=value1
CUR_DATE=2020-10-05
```
19.6.6 Filtering Out Irrelevant Templates From The Interface For Users

The list of kernel templates that are available for users of the Jupyter Kernel Creator can be modified as follows:

In the parent directory of the templates, the file `filter.yaml` describes items called `features`. These can be statically defined or they can represent scripts to be executed. The result of an executed script can be `true` or `false`. If the script code exits with zero, then the result is `true`.

Every kernel template can have the `feature` field, which is a boolean expression that is calculated when the JupyterLab extension is started.

**Example**

```
# cat filter.yaml
...
slurm:
  getter: shell
  timeout: 5
  exec:
    - "source /etc/profile.d/modules.sh"
    - "module load slurm"
    - "sinfo"
...

# cat jupyter-eg-kernel-slurm-py39/meta.yaml
...
features: "slurm"
...
```

In the preceding example, above `jupyter-eg-kernel-slurm-py39` template is shown in the JupyterLab interface if the script in the `exec:` section is able to finish successfully.

It is also possible to define more complicated rules:

**Example**

```
# cat filter.yaml
features:
  kubernetes:
    getter: shell
    timeout: 5
    exec:
      - "source /etc/profile.d/modules.sh"
      - "module load kubernetes"
      - "kubectl get pods"
  k8s-jupyter-operator-installed:
    getter: shell
    timeout: 5
```

![Figure 19.15: Environment variables are available in the running kernel](image)

```python
[1]: import os
[2]: os.environ['VAR1']
[2]: 'value1'
[3]: os.environ['CUR_DATE']
[3]: '2020-10-05'
```
19.7 Kubernetes Jupyter Kernel Operator

exec:
- "source /etc/profile.d/modules.sh"
- "module load kubernetes"
- "kubectl get cmjupyterkernels 2>&1 || true \"
- "| egrep -q 'Error from server \(Forbidden\)'"

k8s-jupyter-operator-enabled:
getter: shell
timeout: 5
exec:
- "source /etc/profile.d/modules.sh"
- "module load kubernetes"
- "kubectl get cmjupyterkernels"
...

# cat jupyter-eg-kernel-k8s-py/meta.yaml
---
display_name: "Python on Kubernetes"
features: "kubernetes and not k8s-jupyter-operator-installed and not k8s-jupyter-operator-enabled"
...

In the preceding example, jupyter-eg-kernel-k8s-py is shown when Kubernetes is installed, and is hidden if Jupyter Kubernetes Operator is available for the user.

The filter.yaml file also supports statically defined features:

Example

features:
  always-enabled:
    getter: static
    default: True
  always-disabled:
    getter: static
    default: False
...

and supports Python code:

Example

python3-available:
getter: python
timeout: 5
exec:
- "import sys"
- "sys.exit(0) if sys.version_info.major == 3 else sys.exit(1)"
...

19.7 Kubernetes Jupyter Kernel Operator

19.7.1 Installing Jupyter Kernel Operator

The Kubernetes Jupyter Kernel Operator can be installed as a part of the cm-kubernetes-setup procedure (section 9.3.2):
The Kubernetes Jupyter Kernel Operator can alternatively be installed later on using the OS package manager and Helm:

```
[root@bright92 ~]# yum install cm-jupyter-kernel-operator -y
[root@bright92 ~]# helm install cm-jupyter-kernel-operator \
    /cm/shared/apps/jupyter-kernel-operator/current/helm/cm-jupyter-kernel-operator-*.tgz
```

The Jupyter kernel Operator can be removed with:

```
[root@bright92 ~]# helm uninstall cm-jupyter-kernel-operator
```

It is recommended to enable the PodSecurityPolicy (PSP, page 463)) for the cluster before allowing a user, for example alice, to create resources in the Kubernetes cluster.

```
[root@bright92 ~]# cm-kubernetes-setup --psp
[root@bright92 ~]# cm-kubernetes-setup --add-user alice --operators cm-jupyter-kernel-operator
```

The Kubernetes Jupyter Kernel Operator Helm chart creates a CRD that can be used in the Kubernetes API. To check the availability of the CRD, the following command can be run:

```
[root@bright92 ~]# su - alice
[alice@bright92 ~]$ module load kubernetes
[alice@bright92 ~]$ kubectl get cmjupyterkernels
```

### 19.7.2 Architecture Overview

The Kubernetes Jupyter Kernel Operator has two main components:

- the Operator itself
- the sidecar. This is attached to every user-defined kernel pod, and communicates with Jupyter Enterprise Gateway, acting as a proxy for the kernel process.

The following is an overview of the kernel setup and pod lifecycle when the user runs the Kubernetes Jupyter Kernel Operator:
1. User initiates creating kernel in JupyterLab.

2. JupyterLab delegates this task to Jupyter Enterprise Gateway (JEG).

3. JEG opens a service TCP/IP socket and creates a CRD in Kubernetes specifying this port.

4. KubeApi notifies Jupyter Kernel Operator about the newly created CRD.

5. Jupyter Kernel Operator creates services, configmaps, secrets.

6. Jupyter Kernel Operator creates pod to run Jupyter kernel based on the specification. The sidecar is added to the kernel pod during this step.

7. The sidecar waits for the connection file created by the kernel. Alternatively, it relies on the connection file created by the operator (if requested), as not all kernels create a connection file.

8. The sidecar runs a proxy to forward kernel communications to JEG (stdin, shell, iopub, etc).

9. The sidecar notifies JEG about connection parameters and handles kernel communications.

10. If JEG disappears, or if communication drops, then the sidecar stops. This causes the kernel operator to get a notification via the KubeApi service.

11. The Kubernetes Jupyter Kernel Operator removes the unneeded pod, service, configmap and secrets. It also tries to gather stdout and std error of the kernel pod for debug purposes.

The pod created in step 6 is heavily customized by the kernel operator. For security reasons, running a process inside the pod must be carried out as an unprivileged user.

For the convenience of the Jupyter user, the UID/GID of the process inside the pod should match the UID/GID of the Jupyter user. If that is not the case, then the files created in the container are inaccessible for the Jupyter user.

To achieve matching UID/GIDs, the operator dynamically creates /etc/passwd and /etc/group files inside the pod and populates them with the data from corresponding templates. At the same time the operator can create a kernel communication file, if requested—some kernels rely on that.
19.7.3 Running Jupyter Kernel Using The Operator

An example of a basic YAML definition for the CMJupyterKernel is:

```
[alice@bright92 ~]$ cat cmjk.yaml
---
apiVersion: apps.brightcomputing.com/v1
kind: CMJupyterKernel
metadata:
  name: cmjk-test
  namespace: alice-restricted
spec:
  username: alice
  uid: 1001
  gid: 1001
  kernel_id: testtesttest
  homedir: /home/alice
pod:
  volumes:
    - name: homedir
      hostPath:
        path: /home/alice
        type: DirectoryOrCreate
  containers:
    - name: kernel
      image: jupyter/datascience-notebook
      command:
        - "python"
      args:
        - "-m"
        - "ipykernel_launcher"
        - "-f"
        - "/var/tmp/kernel-parm.json"
      workingDir: /home/alice
      securityContext:
        allowPrivilegeEscalation: false
        privileged: false
        runAsNonRoot: true
        runAsUser: 1001
        runAsGroup: 1001
      volumeMounts:
        - name: homedir
          mountPath: /home/alice
```

This can be submitted, but the operator removes it in approximately 1 minute:

```
[alice@bright92 ~]$ module load kubernetes
[alice@bright92 ~]$ kubectl apply -f cmjk.yaml
```

The logs of the operator can be checked for debug purposes:

```
[root@bright92 ~]# module load kubernetes
[root@bright92 ~]# kubectl logs \
  -n cm-jupyter-kernel-operator-system \
  -l control-plane=controller-manager \
  --tail -1 \ 
  -c manager
```
This indicates that the sidecar was stopped because there was no connection from Jupyter Enterprise Gateway to the kernel. This is expected, since the kernel has been run manually, and not using Jupyter. After the sidecar shutdown, the kube-api server notifies the operator, which, in turn, removes objects such as CMJupyterKernel, pods, and services.

### 19.7.4 Kernel Operator Tunables

<table>
<thead>
<tr>
<th>Option</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>kernel_id</td>
<td>Kernel identifier (random UUID) given by Jupyter server</td>
</tr>
<tr>
<td>username</td>
<td>Name of the user</td>
</tr>
<tr>
<td>uid, gid, homedir, usershell</td>
<td>UID, GID, home directory and default shell of the user</td>
</tr>
<tr>
<td>image_os_flavor</td>
<td>Defines template of /etc/passwd and /etc/group files, where uid, gid, homedir, and usershell will be added. Could be one of ubuntu1804, rhel7, rhel8, ubuntu2004, sles12, sles15.</td>
</tr>
<tr>
<td>etc_passwd, etc_group</td>
<td>Custom content of the /etc/passwd or /etc/group, if necessary.</td>
</tr>
<tr>
<td>sidecar_command, sidecar_args</td>
<td>Commands and arguments to run the sidecar. By default empty. Most of the arguments for the sidecar are passed via environment variables (section 19.7.5).</td>
</tr>
<tr>
<td>kernel_connection_file_path</td>
<td>Where to expect to find kernel connection file. Default: /var/tmp/kernel-parm.json</td>
</tr>
</tbody>
</table>
The Jupyter Notebook Environment Integration

---

**Option** | **Description**
---|---
create_connection_file\(^O\) | Does the operator need to create and populate kernel connection file before the pod starts? Default: false

spark_pod_template_path\(^O\), spark_pod_template\(^O\) | Options to store or override the Spark executor template

pod\(^R\) | Kubernetes Pod definition

service\(^O\) | Kubernetes Service definition

Legend:
O: Optional
R: Required

---

19.7.5 Sidecar Arguments And Environment Variables

**Sidecar Arguments**

A timeout can be set as an argument for the sidecar.

- `--timeout`: Defines how long, in seconds, that the sidecar waits for the Jupyter Enterprise Gateway proxy to connect before shutdown. Default: 60

**Environment Variables**

The following environment variables can be used by the sidecar:

**Table 19.4: Environment Variables For The Sidecar**

<table>
<thead>
<tr>
<th>Environment Variable</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>CMJK_CONNECTION_FILE</td>
<td>Path to find a connection file. The sidecar uses the file to establish a connection to the kernel and to pass data between Jupyter Enterprise Gateway and the kernel. Default: /var/tmp/kernel-param.json.</td>
</tr>
<tr>
<td>CMJK_KERNEL_ID</td>
<td>Unique identifier of the kernel. Usually the UUID in table 19.3.</td>
</tr>
<tr>
<td>CMJK_SHELL_PORT</td>
<td>Proxy port to open to forward shell communication. Default: 5001.</td>
</tr>
</tbody>
</table>

---

...continues
<table>
<thead>
<tr>
<th>Environment Variable</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>CMJK_IOPUB_PORT</td>
<td>Proxy port to open to forward iopub communication. Default: 5002.</td>
</tr>
<tr>
<td>CMJK_STDIN_PORT</td>
<td>Proxy port to open to forward stdin communication. Default: 5003.</td>
</tr>
<tr>
<td>CMJK_CONTROL_PORT</td>
<td>Proxy port to open to forward control communication. Default: 5004.</td>
</tr>
<tr>
<td>CMJK_HB_PORT</td>
<td>Proxy port to open to forward heartbeat communication. Default: 5005.</td>
</tr>
<tr>
<td>CMJK_COMM_PORT</td>
<td>Proxy port to open to forward comm communication. Default: 5006.</td>
</tr>
</tbody>
</table>

### 19.7.6 Running Spark-based Kernels In Jupyter Kernel Operator

Jupyter integration for the cluster manager provides a kernel template (`jupyter-eg-kernel-k8s-cmkop-py-spark`) and a sample container image (`brightcomputing/jupyter-kernel-sample:k8s-spark-py39-2.0.0`) to run Jupyter kernels in a Spark environment. The image can be altered or created from scratch based on the scripts provided in `/cm/shared/examples/jupyter/kubernetes-kernel-image-spark-py39/`.

The Jupyter kernel is not run directly. Instead, the kernel process is run and controlled by the `spark-submit` executable inside the container.

Jupyter Kernel Operator alters the provided image based on the CRD definition.

Spark-specific tunables are `spark_pod_template_path` and `spark_pod_template`. The Kernel Operator creates a file inside of the Spark driver Pod and puts the content of `spark_pod_template` in it. After that, `spark-submit` uses this file, via the `--spark.kubernetes.executor.podTemplateFile` configuration option, to create executor pods.

### 19.7.7 Example: Creating An R kernel

The Jupyter Kernel Operator can be used out-of-the-box to support more kernels. For example, an R kernel can be added.


The default entry point cannot be used as that would start Jupyter notebook, while the aim for this section is to use the kernel only.

Some exploratory investigation should reveal the command to start the kernel:

The pod can be run interactively in a Jupyter notebook terminal by a user:

```bash
ekubectl run -i --tty testnotebook --image=jupyter/r-notebook --restart=Never -- bash
```

The kernel specifications can then be investigated:

```bash
jupyter-kernelspec list
```

Available kernels:

- `ir` /opt/conda/share/jupyter/kernels/ir
- `python3` /opt/conda/share/jupyter/kernels/python3

The `ir` kernel is what is of interest here. The command line that is used to start the kernel can be found:

```bash
cat /opt/conda/share/jupyter/kernels/ir/kernel.json
```

```
"argv": ["R", "--slave", ",e", "IRKernel::main()", "--args", "connection_file"]
```
Based on this information the Jupyter Kernel Operator CRD can be created for the user:

```yaml
cat cmjk-ir.yaml
---
apiVersion: apps.brightcomputing.com/v1
kind: CMJupyterKernel
metadata:
  name: cmjk-test
  namespace: alice-restricted
spec:
  username: alice
  uid: 1001
  gid: 1001
  kernel_id: testtesttest
  homedir: /home/alice
  create_connection_file: true # R kernel expects connection file be created
pod:
  volumes:
    - name: homedir
      hostPath:
        path: /home/alice
        type: DirectoryOrCreate
  containers:
    - name: kernel
      image: jupyter/r-notebook # image
      command:
        - "R"
      args:
        - "--slave"
        - "-e"
        - "IRkernel::main()"
        - "--args"
        - "-/var/tmp/kernel-parm.json" # we have static connection file
      workingDir: /home/alice
      securityContext:
        allowPrivilegeEscalation: false
        privileged: false
        runAsNonRoot: true
        runAsUser: 1001
        runAsGroup: 1001
  volumeMounts:
    - name: homedir
      mountPath: /home/alice
```

There are several changes from the previous (section 19.7.3) YAML, and from the IR command line:

- `create_connection_file: true` # R kernel expects connection file be created

If this is not specified then the kernel complains with the following message during startup:

```
kernel:  cannot open file '/var/tmp/kernel-parm.json': No such file or directory
```

This means that the kernel expected this file to be created before the start.
• image: jupyter/r-notebook

Another image needs to be used.

• args: ...

"/var/tmp/kernel-parm.json"

The spec file has a fixed path and name, instead of "connection_file" as in kernel.json earlier.

The resulting cmjk-ir.yaml file can be submitted to Kubernetes, but it will be removed by the operator after one minute, as it is not being started from the Jupyter Enterprise Gateway.

The next step is to create a kernel template. The Python kernel can be used as a reference:

```
cd /cm/shared/apps/jupyter/current
cd lib/python*/site-packages/cm_jupyter_kernel_creator/kerneltemplates
cp -pr jupyter-eg-kernel-k8s-cmjkop-py jupyter-eg-kernel-k8s-cmjkop-r
```

The files meta.yaml, kernel.json, and templates/cmjk.yaml.j2 need to be changed in order to allow the correct image and command:

```
vim jupyter-eg-kernel-k8s-cmjkop-r/meta.yaml
vim jupyter-eg-kernel-k8s-cmjkop-r/kernel.json.j2
vim templates/cmjk.yaml.j2
```

The changes that are applied should look similar to the following:

```
diff -u jupyter-eg-kernel-k8s-cmjkop-py/kernel.json.j2 jupyter-eg-kernel-k8s-cmjkop-r/kernel.json.j2
+++ jupyter-eg-kernel-k8s-cmjkop-r/kernel.json.j2 2022-02-16 12:22:23.610382929 +0100
@@ -15,8 +15,8 @@
   },
   "argv": [
-   "python",
-   "-m", "ipykernel_launcher",
-   "-F", "/var/tmp/kernel-parm.json"
+   "R",
+   "--slave", "-e", "IRkernel::main()

 differed

diff -u jupyter-eg-kernel-k8s-cmjkop-py/meta.yaml jupyter-eg-kernel-k8s-cmjkop-r/meta.yaml
+++ jupyter-eg-kernel-k8s-cmjkop-r/meta.yaml 2022-02-16 12:20:57.500974886 +0100
@@ -1,5 +1,5 @@
-    display_name: "Python on Kubernetes Operator"
+    display_name: "R on Kubernetes Operator"
    features: ["k8s-jupyter-operator-enabled"]
    parameters:
      display_name:
```

```
exec:
- - echo "Python on Kubernetes Operator $(date +%y%m%d%H%M%S)"
+ - echo "R on Kubernetes Operator $(date +%y%m%d%H%M%S)"

display_name: "Display name of the kernel"
k8s_env_module:
  type: str
@@ -20,9 +20,9 @@
definition:
  getter: static
  default:
- - "jupyter/datascience-notebook"
+ - "jupyter/r-notebook"
  values:
- - "jupyter/datascience-notebook"
+ - "jupyter/r-notebook"
  display_name: "Image to run"
  limits:
    max_len: 1

After instantiating a kernel spec from the template, the R kernel is ready to use:
Figure 19.17: Jupyter Kernel Creator. Creating the IR kernel spec
19.8 Jupyter VNC Extension

If VNC is allowed by the cluster administrator, then the Jupyter environment configured by Bright can be used to start and control remote desktops via VNC with the Jupyter VNC extension.

Several kernels created from Bright’s templates are capable of running VNC sessions so that users can run GUI applications. In order to do so, the cluster nodes where kernels are executed must support VNC. It is outside of the scope of this section to provide guidance on how to install and configure VNC on different Linux distributions. However, it is important to note that Jupyter kernels usually work with several types of VNC client/server.

Bright’s default choice for VNC is TigerVNC, a high-performance, platform-neutral implementation of VNC.

For example, the administrator can install the VNC server on the head node as follows:

Example

[root@bright92 ]# yum install tigervnc-server #on RHEL 7 and 8
[root@bright92 ]# zypper install tigervnc #on SLES 15
[root@bright92 ]# apt-get install tightvncserver #on Ubuntu 18.04 and 20.04

The preceding command must be applied to all software images that are used to run Jupyter kernels for VNC.

Users can start a VNC session with the button added by Jupyter VNC (figure 19.19). Additional VNC parameters can be optionally specified.
If VNC is available and correctly configured on the node where the kernel is running, then a new tab is automatically created by Jupyter VNC containing the new session (figure 19.20). A user can now freely interact in JupyterLab both with the notebook and with the desktop environment.

To provide a user-friendly experience, Jupyter VNC also allows the graphical viewport to be resized, so that the desktop application can run full-screen (figure 19.21).
Once the VNC session is correctly started and the new JupyterLab tab has been created, Jupyter VNC automatically exports the DISPLAY environment variable to the running notebook (figure 19.22). Doing so means that any application or library running in the notebook can make use of the freshly created desktop environment. An example of such a library is OpenAI Gym, a toolkit for developing and comparing reinforcement learning algorithms, that is distributed by Bright.

Among the examples distributed by Bright (section 19.3), a notebook running PyTorch in the OpenAI Gym CartPole environment can be found. If executed after a VNC session has been started, a user can then observe the model being trained in real time in the graphical environment.

19.9 Jupyter WLM Magic Extension

In the Jupyter environment configured by Bright, the Jupyter WLM Magic extension can be used to schedule workload manager jobs from notebooks.

The Jupyter WLM Magic extension is an IPython extension. It is designed to improve the capabilities
of Jupyter’s default Python 3 kernel, which runs on the login node.

The Jupyter WLM Magic extension should therefore not be used from kernels running on compute nodes, such as those typically created with Bright’s Jupyter Kernel Creator extension (section 19.5), and submitted via Jupyter Enterprise Gateway. Indeed, compute nodes running these kernels are often incapable of starting workload manager jobs in many default WLM configurations.

Jupyter WLM Magic extension makes it possible for users to programmatically submit WLM jobs, and then interact with their results. This can be done while using the Python programming language and its libraries, which are available in the notebook.

Users submit jobs and check their progress from the login node. The actual computation is distributed by the underlying workload manager across compute nodes, which means that server resources are spared.

Jupyter WLM Magic commands are available in the IPython kernel as magic functions (https://ipython.readthedocs.io/en/stable/interactive/tutorial.html#magic-functions). A new line magic (%) and a new cell magic (%%) are now added in the kernel, according to the workload manager:

- Platform LSF: %lsf_job and %%lsf_job
- PBS Pro: %pbspro_job and %%pbspro_job
- Slurm: %slurm_job and %%slurm_job
- UGE: %uge_job and %%uge_job

A user can list the magic functions in the kernel to see if they are available, with Jupyter’s builtin command %lsmagic (https://ipython.readthedocs.io/en/stable/interactive/magics.html#magic-lsmagic):

Example

In [ ]: %lsmagic
Out [ ]: root:
line:
  automagic:"AutoMagics"
  autocall:"AutoMagics"
  [...]  
  slurm_job:"SLURMMagic"
  pbspro_job:"PBSProMagic"
  lsf_job:"LSFMagic"
  uge_job:"UGE_magic"

The magic functions introduced by this Bright extension share a similar syntax. For convenience, Slurm is used as an example in this section. However, the same instructions are valid for the other WLMs.

Users can check which options are available for a WLM function with the line magic helper:

Example
### The Jupyter Notebook Environment Integration

In 

```bash
In []: %slurm_job --help
```

Out 

```bash
usage: %slurm_job [-h] [--module MODULE] [--module-load-cmd MODULE_LOAD_CMD]
                        [--shell SHELL] [--submit-command SUBMIT_COMMAND]
                        [--cancel-command CANCEL_COMMAND]
                        [--control-command CONTROL_COMMAND]
                        [--stdout-file STDOUT_FILE] [--stderr-file STDERR_FILE]
                        [--preamble PREAMBLE] [--timeout TIMEOUT]
                        [--check-condition-var CHECK_CONDITION_VAR]
                        [--job-id-var JOB_ID_VAR]
                        [--stdout-file-var STDOUT_FILE_VAR]
                        [--stderr-file-var STDERR_FILE_VAR] [--dont-wait]
                        [--write-updates WRITE_UPDATES]
                        [--check-status-every CHECK_STATUS_EVERY]
```

optional arguments:

- `-h, --help show this help message and exit`

```
...]
```

Line magic functions are typically used to set options with a global scope in the notebook. By doing so, a user will not need to specify the same option every time a job will be submitted via cell magic. For example, if two Slurm instances are deployed on the cluster and their associated environment modules are `slurm-primary` and `slurm-secondary`, a user could run the following line magic once to configure the Jupyter WLM Magic extension to always use the second deployment:

**Example**

```bash
Example

In []: %slurm_job --module slurm-secondary
```

Out 

```
```

Now, jobs will always be submitted to `slurm-secondary`. This is more convenient than repeatedly defining the same module option for every cell magic upon scheduling a job:

**Example**

```bash
Example

In []: %%slurm_job --module slurm-secondary
```

Out 

```
<WLM JOB DEFINITION>
```

```
```

```
```

It should be noted that line magic functions cannot be used to submit WLM jobs. Cell magic functions have to be used instead.

A well-defined cell contains the WLM cell magic function provided by the extension, followed by the traditional job definition. For example, a simple MPI job running on two nodes can be submitted to Slurm by defining and running this cell:

**Example**

```bash
Example

In []: %%%slurm_job --module slurm-secondary
```

Out 

```
#SBATCH -J mpi-job-example
#SBATCH -N 2
module load openmpi
mpirun hostname
```

```
COMPLETED
```

STDOUT file content: `/home/demo/.jupyter/wlm_magic/slurm-1.out`
Users can take advantage of the Jupyter WLM Magic extension to store some information into Python variables about the job being submitted. The information could be the ID or the output file name, for example. Users can then later programmatically interact with them in Python. This feature is convenient when a user wants to, for example, programmatically carry out new actions depending on the job output:

**Example**

```
In []: %%slurm_job --job-id-var my_job_id --stdout-file-var my_job_out
   #SBATCH -J mpi-job-example
   #SBATCH -N 2
   module load openmpi
   mpirun hostname
Out []: COMPLETED
   STDOUT file content: /home/demo/.jupyter/wlm_magic/slurm-2.out
   node001
   node001
   node002
   node002

In []: print(f"Job id {my_job_id} was written to {my_job_out}")
   print(f"Output lines: {open(my_job_out).readlines()}")
Out []: Job id 2 was written to /home/demo/.jupyter/wlm_magic/slurm-2.out
   Output lines: ['node001\n', 'node001\n', 'node002\n', 'node002\n']
```

Users can also exploit Python variables to define the behavior of the Jupyter WLM Magic extension. For example, they can define a Python boolean variable to submit a WLM job only if a condition is true:

**Example**

```
In []: run_job = 1 == 2
Out []:

In []: %%slurm_job --check-condition-var run_job
   #SBATCH -J mpi-job-example
   #SBATCH -N 2
   module load openmpi
   mpirun hostname
Out []: Variable run_job is 'False'. Skipping submit.
```

## 19.10 Jupyter Environment Removal

Before removing Jupyter, the administrator should ensure that all kernels have been halted, and that no user is still logged onto the web interface. Stopping the `cm-jupyterhub` service with users that are still logged in, or with running kernels, has undefined behavior.

To remove Jupyter, the script `cm-jupyter-setup` must be run, either in interactive mode, or with the option `--remove`.

Removing Jupyter does not remove or affect Kubernetes or WLM deployments.

For a more complete cleanup, the following packages must be manually removed from the nodes involved in the Jupyter deployment: `cm-jupyter`, `cm-jupyter-local`, and `cm-npm-configurable-http-proxy`. 
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 (section A.3). These files should not be confused with configuration files that are merely installed (section A.4).

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.1 System Configuration Files Created Or Modified By CMDaemon On Head Nodes

The more important head node files that are managed by CMDaemon are listed here for a plain installation on the various distributions. The list excludes the files managed by CMDaemon in the software images used for regular nodes (section A.2), which are by default located on the head node under `/cm/images/`:

Files generated or modified automatically on the head node by CMDaemon In RHEL and derivatives, Ubuntu 18.04, Ubuntu 20.04, SLES12, SLES15

<table>
<thead>
<tr>
<th>File</th>
<th>Part</th>
<th>Comment</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>/cm/local/apps/openldap/etc/slapd.conf</code></td>
<td>Section</td>
<td></td>
</tr>
<tr>
<td><code>/cm/local/apps/cluster-tools/scripts/finalise-slave.&lt;DISTRO&gt;</code></td>
<td>Section</td>
<td><code>&lt;DISTRO&gt;</code> can be one of CENTOS7 ROCKY8 RHEL7 RHEL8 SLES12 SLES12SP5 SLES15 SLES15SP2 UBUNTU1804 UBUNTU2004</td>
</tr>
</tbody>
</table>

...continues
...continued

<table>
<thead>
<tr>
<th>File</th>
<th>Part</th>
<th>Comment</th>
</tr>
</thead>
<tbody>
<tr>
<td>/cm/local/apps/&lt;PBS&gt;/var/cm/cm-pbs.conf</td>
<td>Section</td>
<td>&lt;PBS&gt; can be one of openpbs pbspro</td>
</tr>
<tr>
<td>/cm/node-installer/etc/sysconfig/clock</td>
<td>Section</td>
<td></td>
</tr>
<tr>
<td>/etc/aliases</td>
<td>Section</td>
<td></td>
</tr>
<tr>
<td>/etc/bind/named.conf</td>
<td>Entire file</td>
<td>Ubuntu only. For zone additions use /etc/bind/named.conf.include. For options additions, use /etc/bind/named.conf.global.options.include.</td>
</tr>
<tr>
<td>/etc/chrony.conf</td>
<td>Section</td>
<td>RHEL8 and derivatives only</td>
</tr>
<tr>
<td>/etc/dhcpd.conf</td>
<td>Entire file</td>
<td></td>
</tr>
<tr>
<td>/etc/dhcpd.internalnet.conf</td>
<td>Section</td>
<td>RHEL7,8 and derivatives, Ubuntu 18.04, Ubuntu 20.04</td>
</tr>
<tr>
<td>/etc/dhcpd.internalnet.conf</td>
<td>Entire file</td>
<td>For internal networks other than internalnet, corresponding files are generated if node booting (table 3.2.1) is enabled</td>
</tr>
<tr>
<td>/etc/exports</td>
<td>Section</td>
<td></td>
</tr>
<tr>
<td>/etc/fstab</td>
<td>Section</td>
<td></td>
</tr>
<tr>
<td>/etc/genders</td>
<td>Section</td>
<td></td>
</tr>
<tr>
<td>/etc/hosts</td>
<td>Section</td>
<td></td>
</tr>
<tr>
<td>/etc/localhost</td>
<td>Symlink</td>
<td></td>
</tr>
<tr>
<td>/etc/logrotate.d/slurm</td>
<td>Entire file</td>
<td></td>
</tr>
<tr>
<td>/etc/logrotate.d/slurmd</td>
<td>Entire file</td>
<td></td>
</tr>
<tr>
<td>/etc/named.conf</td>
<td>Entire file</td>
<td>Non-Ubuntu distributions only. For zone additions use /etc/named.conf.include. For options additions, use /etc/named.conf.global.options.include.</td>
</tr>
<tr>
<td>/etc/ntpd.conf</td>
<td>Section</td>
<td>RHEL7 and derivatives only</td>
</tr>
<tr>
<td>/etc/ntpd/step-tickers</td>
<td>Section</td>
<td>RHEL7 and derivatives only</td>
</tr>
<tr>
<td>/etc/postfix/canonical</td>
<td>Section</td>
<td></td>
</tr>
<tr>
<td>/etc/postfix/generic</td>
<td>Section</td>
<td></td>
</tr>
<tr>
<td>/etc/postfix/main.cf</td>
<td>Section</td>
<td></td>
</tr>
<tr>
<td>/etc/resolv.conf</td>
<td>Section</td>
<td></td>
</tr>
</tbody>
</table>

...continues
### 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>`:

<table>
<thead>
<tr>
<th>File</th>
<th>Part</th>
<th>Comment</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>/etc/shorewall6/interfaces</code></td>
<td>Section</td>
<td></td>
</tr>
<tr>
<td><code>/etc/shorewall6/policy</code></td>
<td>Section</td>
<td></td>
</tr>
<tr>
<td><code>/etc/shorewall6/rules</code></td>
<td>Section</td>
<td></td>
</tr>
<tr>
<td><code>/etc/shorewall6/snat</code></td>
<td>Section</td>
<td></td>
</tr>
<tr>
<td><code>/etc/shorewall6/zones</code></td>
<td>Section</td>
<td></td>
</tr>
<tr>
<td><code>/etc/shorewall/interfaces</code></td>
<td>Section</td>
<td></td>
</tr>
<tr>
<td><code>/etc/shorewall/netmap</code></td>
<td>Section</td>
<td></td>
</tr>
<tr>
<td><code>/etc/shorewall/policy</code></td>
<td>Section</td>
<td></td>
</tr>
<tr>
<td><code>/etc/shorewall/rules</code></td>
<td>Section</td>
<td></td>
</tr>
<tr>
<td><code>/etc/shorewall/snat</code></td>
<td>Section</td>
<td></td>
</tr>
<tr>
<td><code>/etc/shorewall/zones</code></td>
<td>Section</td>
<td></td>
</tr>
<tr>
<td><code>/etc/snmp/snmptrapd.conf</code></td>
<td>Section</td>
<td></td>
</tr>
<tr>
<td><code>/etc/sysconfig/bmccfg</code></td>
<td>Entire file</td>
<td>BMC configuration</td>
</tr>
<tr>
<td><code>/etc/sysconfig/clock</code></td>
<td>Section</td>
<td></td>
</tr>
<tr>
<td><code>/etc/sysconfig/dhcpd</code></td>
<td>Entire file</td>
<td></td>
</tr>
<tr>
<td><code>/etc/sysconfig/network-scripts/ifcfg-*</code></td>
<td>Section</td>
<td></td>
</tr>
<tr>
<td><code>/tftpboot/mtu.conf</code></td>
<td>Entire file</td>
<td>Bright Computing configuration</td>
</tr>
<tr>
<td><code>/tftpboot/pixelinux.cfg/category.*.*</code></td>
<td>Entire file</td>
<td>Bright Computing configuration</td>
</tr>
<tr>
<td><code>/var/lib/named/*.*.zone</code></td>
<td>Entire file</td>
<td>For SLES distributions only. For custom additions use <code>/var/lib/named/*.*.zone.include</code></td>
</tr>
<tr>
<td><code>/var/named/*.*.zone</code></td>
<td>Entire file</td>
<td>For non-SLES distributions only. For custom additions use <code>/var/named/*.*.zone.include</code></td>
</tr>
</tbody>
</table>

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.
System configuration files created or modified by CMDaemon in software images in RHEL 7/8 and derivatives, SLES 12, SLES 15, Ubuntu 18.04, Ubuntu 20.04

<table>
<thead>
<tr>
<th>File</th>
<th>Part</th>
<th>Comment</th>
</tr>
</thead>
<tbody>
<tr>
<td>/etc/aliases</td>
<td>Section</td>
<td></td>
</tr>
<tr>
<td>/etc/hosts</td>
<td>Section</td>
<td></td>
</tr>
<tr>
<td>/etc/modprobe.d/bright-cmdaemon.conf</td>
<td>Section</td>
<td></td>
</tr>
<tr>
<td>/etc/mkinitrd_cm.conf</td>
<td>Section</td>
<td>Not for Ubuntu</td>
</tr>
<tr>
<td>/etc/initramfs-tools/modules</td>
<td>Section</td>
<td>Only for Ubuntu</td>
</tr>
<tr>
<td>/etc/modprobe.d/bright-cmdaemon.conf</td>
<td>Section</td>
<td></td>
</tr>
<tr>
<td>/etc/postfix/main.cf</td>
<td>Section</td>
<td></td>
</tr>
<tr>
<td>/etc/rsyslog.conf</td>
<td>Section</td>
<td></td>
</tr>
<tr>
<td>/etc/securetty</td>
<td>Section</td>
<td></td>
</tr>
<tr>
<td>/etc/shorewall/rules</td>
<td>Section</td>
<td></td>
</tr>
<tr>
<td>/etc/shorewall/snat</td>
<td>Section</td>
<td></td>
</tr>
<tr>
<td>/etc/shorewall6/rules</td>
<td>Section</td>
<td></td>
</tr>
<tr>
<td>/etc/shorewall6/snat</td>
<td>Section</td>
<td></td>
</tr>
<tr>
<td>/etc/sysconfig/clock</td>
<td>Section</td>
<td></td>
</tr>
</tbody>
</table>

A.3 Files Created On Regular Nodes By The Node-Installer

The list of files on the regular node that are generated or modified by the node-installer can be viewed in the logs at /var/log/modified-by-node-installer.log on the node. For a default installation in RHEL 7/8 and derivatives, the list is as shown in the following table:
A.4 Files Not Generated, But Installed In RHEL And Derivatives

This appendix (Appendix A) is mainly about generated configuration files. This section (A.4) 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@bright92 ~]# 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 NVIDIA Bright 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:

---

System configuration files created or modified on regular nodes by the node-installer in RHEL7/8 and derivatives

<table>
<thead>
<tr>
<th>File</th>
<th>Part</th>
<th>Comment</th>
</tr>
</thead>
<tbody>
<tr>
<td>/etc/sysconfig/network-scripts/ifcfg-*</td>
<td>Entire file</td>
<td>not ifcfg-log</td>
</tr>
<tr>
<td>/etc/sysconfig/network</td>
<td>Section</td>
<td></td>
</tr>
<tr>
<td>/etc/resolv.conf</td>
<td>Entire file</td>
<td></td>
</tr>
<tr>
<td>/etc/hosts</td>
<td>Section</td>
<td></td>
</tr>
<tr>
<td>/etc/postfix/main.cf</td>
<td>Section</td>
<td></td>
</tr>
<tr>
<td>/etc/chrony.conf</td>
<td>Entire file</td>
<td>RHEL8 and derivatives only</td>
</tr>
<tr>
<td>/etc/ntp.conf</td>
<td>Entire file</td>
<td>RHEL7 and derivatives only</td>
</tr>
<tr>
<td>/etc/ntp/step-tickers</td>
<td>Entire file</td>
<td>RHEL7 and derivatives only</td>
</tr>
<tr>
<td>/etc/hostname</td>
<td>Section</td>
<td></td>
</tr>
<tr>
<td>/cm/local/apps/cmd/etc/cert.pem</td>
<td>Section</td>
<td></td>
</tr>
<tr>
<td>/cm/local/apps/cmd/etc/cert.key</td>
<td>Section</td>
<td></td>
</tr>
<tr>
<td>/cm/local/apps/openldap/etc/certs/ldap.pem</td>
<td>Section</td>
<td></td>
</tr>
<tr>
<td>/cm/local/apps/openldap/etc/certs/ldap.key</td>
<td>Section</td>
<td></td>
</tr>
<tr>
<td>/var/spool/cmd/disks.xml</td>
<td>Section</td>
<td></td>
</tr>
<tr>
<td>/cm/local/apps/cmd/etc/cluster.pem</td>
<td>Section</td>
<td></td>
</tr>
<tr>
<td>/var/log/rsyncd.log</td>
<td>Section</td>
<td></td>
</tr>
<tr>
<td>/var/log/node-installer</td>
<td>Section</td>
<td></td>
</tr>
<tr>
<td>/var/log/modified-by-node-installer.log</td>
<td>Section</td>
<td></td>
</tr>
</tbody>
</table>
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:

```
$ repoquery -l $(repoquery -a | grep -F -v _cm9.2) | grep nsswitch.conf$
```

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-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.3 of this appendix.

Similarly, looking through the output for the less specific case:

```
"# yum whatprovides */etc/nsswitch.conf
...
"cm-ldap-client\n"
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} $(dpkg -S "nsswitch.conf" | cut -f1 -d:) | grep cm9.1
cm-config-ldap-client-master 9.1-131-cm9.1
```

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. 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 '%{FILENAME}\n%{FILEFLAGS:fflags}\n' glibc | egrep '[:space:]\n.*\\.(c|n).*$' | sort -u
```

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:

---

1This 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 12.3.2), thereby avoiding the overwriting of that file by the excluded package.

A request to change the package build flag may be sent to Bright Computing if the preceding suggested options are unworkable.
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 857.

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 Bright View and cmsh,
except for when CMDaemon is started up from init.

**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 authentica-
tion 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 13.10.3) 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 13.10.3) specifies the path to the file that contains regular expressions used to filter out incoming messages on the event-bucket.

**LDAPHost directive**

**Syntax:** LDAPHost = hostname  
**Default:** LDAPHost = "localhost"

The **LDAPHost** directive specifies the hostname of the LDAP server to connect to for user management.

**LDAPUser directive**

**Syntax:** LDAPUser = username  
**Default:** LDAPUser = "root"

The **LDAPUser** directive specifies the username used when connecting to the LDAP server.

**LDAPPass directive**

**Syntax:** LDAPPass = password  
**Default:** LDAPPass = "<random string set during installation>"

The **LDAPPass** directive specifies the password used when connecting to the LDAP server. It can be changed following the procedure described in Appendix I.

**LDAPReadOnlyUser directive**

**Syntax:** LDAPReadOnlyUser = username  
**Default:** LDAPReadOnlyUser = "readonlyroot"

The **LDAPReadOnlyUser** directive specifies the username that will be used when connecting to the LDAP server during LDAP replication. The user is a member of the "rogroup" group, whose members have a read-only access to the whole LDAP directory.

**LDAPReadOnlyPass directive**

**Syntax:** LDAPReadOnlyPass = password  
**Default:** LDAPReadOnlyPass = "<random string set during installation>"

The **LDAPReadOnlyPass** directive specifies the password that will be used when connecting to the LDAP server during LDAP replication.

**LDAPSearchDN directive**

**Syntax:** LDAPSearchDN = dn
Default: `LDAPSearchDN = "dc=cm,dc=cluster"`

The `LDAPSearchDN` directive specifies the Distinguished Name (DN) used when querying the LDAP server.

**LDAPProtocol directive**

Syntax: `LDAPProtocol = ldap|ldaps`

Default: `LDAPProtocol = "ldaps"`

The `LDAPProtocol` directive specifies the LDAP protocol to be used when querying the LDAP server.

**LDAPPort directive**

Syntax: `LDAPPort = number`

Default: `LDAPPort = 636`

The `LDAPPort` directive specifies the port to be used when querying the LDAP server.

**LDAPCACertificate directive**

Syntax: `LDAPCACertificate = filename`

Default: `LDAPCACertificate = "/cm/local/apps/openldap/etc/certs/ca.pem"`

The `LDAPCACertificate` directive specifies the CA certificate to be used when querying the LDAP server.

**LDAPCertificate directive**

Syntax: `LDAPCertificate = filename`

Default: `LDAPCertificate = "/cm/local/apps/openldap/etc/certs/ldap.pem"`

The `LDAPCertificate` directive specifies the LDAP certificate to be used when querying the LDAP server.

**LDAPPrivateKey directive**

Syntax: `LDAPPrivateKey= filename`

Default: `LDAPPrivateKey= "/cm/local/apps/openldap/etc/certs/ldap.key"`

The `LDAPPrivateKey` directive specifies the LDAP key to be used when querying the LDAP server.

**HomeRoot directive**

Syntax: `HomeRoot = path`

Default: `HomeRoot = "/home"`

The `HomeRoot` directive specifies the default user home directory used by CMDaemon. It is used for automatic mounts, exports, and when creating new users.

**DocumentRoot directive**

Syntax: `DocumentRoot = path`
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 Bright View. The connection runs over CMDaemon, which is running over SSL, which means that between Bright 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 enables monitoring plots in Bright View and the User Portal.

**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

CMDaemonAuditorFile = *path to audit.json file*

or

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 Bright 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 853).

Whether events are logged in files or not, events are cached and accessible using `cmsh` or Bright View. The number of events cached by CMDaemon is determined by the parameter `MaxEventHistory` (page 853).

**EventLoggerFile directive**

**Syntax:** EventLoggerFile = filename

**Default:** EventLoggerFile = "/var/spool/cmd/events.log"

The `EventLoggerFile` directive sets where the events seen in the event viewer (section 13.10.1) are logged.

**MaxEventHistory directive**

**Syntax:** AdvancedConfig = {"MaxEventHistory=number", ...}

**Default:** MaxEventHistory=8192

MaxEventHistory is a parameter of the `AdvancedConfig` (page 862) 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 853) 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 863) 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 `csh` or Bright View (section 3.2.1, figure 3.5, table 3.2.1). Setting the `cmd.conf LockDownDhcpd` directive overrides `lockdowndhcpd` values set by `csh` or Bright 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`

Configure the username and password for the BMC interface of the head node and regular nodes automatically. (This should not be confused with the `setupBmc` field of the node-installer configuration file, described in section 5.8.7.)

**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 862) 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 862) 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 862) 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` directive. It is a parameter of the `AdvancedConfig` 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  
- UGE  
- 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**

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 12.7) 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 567.

**Example**

Per node:

`frozenFilesPerNode = "*:localhost/etc/ntp.conf", "node003:localhost/etc/hosts"`

Here, the `*` wildcard means that no restriction is set. Setting `node003` means that only node003 is frozen.

**Example**

Per category:

`frozenFilesPerCategory = "mycategory:localhost/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:

```plaintext
mydomain = eth.cluster
```

can often be overridden by a statement later on outside the CMDaemon-maintained part of the file:

```plaintext
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 Bright support.
• 2 allows violation of the requirements, without validation warnings. This value should never be set except under instructions from Bright support.

**CustomUpdateConfigFileScript directive**

Syntax: `CustomUpdateConfigFileScript = filename`

Default: commented out in the default cm.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/syslog.conf` in Red Hat 5 and variants
- `/etc/rsyslog.conf` in Red Hat 6 and variants
- `/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 862) 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:

<table>
<thead>
<tr>
<th>ResolveToExternalName Directive Effects</th>
</tr>
</thead>
<tbody>
<tr>
<td>on simple head, resolving: master head</td>
</tr>
<tr>
<td>----------------------------------------</td>
</tr>
<tr>
<td>ResolveToExternalName = False</td>
</tr>
<tr>
<td>I I I I I I I I I I I I I I I I I I</td>
</tr>
<tr>
<td>I I I I I I I I I I I I I I I I I I</td>
</tr>
<tr>
<td>ResolveToExternalName = True</td>
</tr>
<tr>
<td>E E E E E E E E E E E E E E E E E E</td>
</tr>
<tr>
<td>E E E E E E E E E E E E E E E E E E</td>
</tr>
</tbody>
</table>

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.brightcomputing.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 862) directive.

ResolveMasterToExternalName can only be used if ResolveToExternalName (page 860) 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 862) directive.

ResolveMasterToExternalDomainName can only be used if ResolveToExternalName (page 860) 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:

<table>
<thead>
<tr>
<th>on simple head, resolving:</th>
<th>on failover head, resolving:</th>
<th>on regular node, resolving:</th>
<th>Using the DNS?</th>
</tr>
</thead>
<tbody>
<tr>
<td>master head</td>
<td>master head1 head2</td>
<td>master head(s)</td>
<td></td>
</tr>
<tr>
<td><strong>ResolveToExternalName = False</strong></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>I</td>
<td>I</td>
<td>I</td>
<td>No</td>
</tr>
<tr>
<td>I</td>
<td>I</td>
<td>I</td>
<td>Yes</td>
</tr>
<tr>
<td><strong>ResolveToExternalName = True</strong></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>E</td>
<td>E</td>
<td>E</td>
<td>No</td>
</tr>
<tr>
<td>E</td>
<td>I</td>
<td>E</td>
<td>Yes</td>
</tr>
</tbody>
</table>

Key:  
I: resolves to internal IP address of head
E: resolves to external IP address of head
**DisableLua directive**

**Syntax:** DisableLua = true|false

**Default:** DisableLua = false

The value of the DisableLua directive determines if Lua code (section L.6) used in monitoring expressions can be executed.

**AdvancedConfig directive**

**Syntax:** AdvancedConfig = { "<key1>=<value1>", "<key2>=<value2>", ... }

**Default:** Commented out in the default cmd.conf file

The AdvancedConfig directive is not part of the standard directives. It takes a set of key/value pairs as parameters, with each key/value pair allowing a particular functionality, and is quite normal in that respect. However, the functionality of a parameter to this directive is often applicable only under restricted conditions, or for non-standard configurations. The AdvancedConfig parameters are therefore generally not recommended, nor are they documented, for use by the administrator.

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@bright92 ~]# 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@bright92 ~]# cm-manipulate-advanced-config.py -s

 VirtualCluster=1
 key8=value8

A key/value pair can be removed by specifying its key with the -r|--remove option:

[root@bright92 ~]# cm-manipulate-advanced-config.py -r key8

Updated: /cm/local/apps/cmd/etc/cmd.conf

[root@bright92 ~]# cm-manipulate-advanced-config.py -s

 VirtualCluster=1

The utility can be used on cmd.conf in node images too, using the -i|--image option:

[root@bright92 ~]# 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 854).

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:

```plaintext
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 595) 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 862) directive. Its value determines the path of the directory in which monitoring data is saved (section 17.8).

MaxServiceFailureCount directive
Syntax: AdvancedConfig = {"MaxServiceFailureCount=number", ...}
Default: Implicit value: "MaxServiceFailureCount=10"

MaxServiceFailureCount is a parameter of the AdvancedConfig (page 862) directive. Its value determines the number of times a service failure event is logged (page 123). 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 862) 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 864), 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 123) on the fhgfs-client from within CMDaemon, as follows:

```
[bright92->category[default]->services[fhgfs-client]]% reset fhgfs-client
Successfully reset service fhgfs-client on: node001,node002
[bright92->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 862) 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 862) directive. It defines the number of seconds until the Bright View connection to CMDaemon times out, if there is no user activity at the Bright View client side.

**DHCPMaxleaseTime directive**

**Syntax:** `AdvancedConfig = {"DHCPMaxleaseTime=number", ...}

**Default:** client default

DHCPMaxleaseTime is a parameter of the AdvancedConfig (page 862) 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.

**SlurmDisableAccountingParsing directive**

**Syntax:** `AdvancedConfig = {"SlurmDisableAccountingParsing=0|1", ...}

**Default:** 0

SlurmDisableAccountingParsing is a parameter of the AdvancedConfig (page 862) directive. If set to 1, it disables collection of accounting information for the Slurm workload manager.

**JobsSamplingMetricsInterval directive**

**Syntax:** `AdvancedConfig = {"JobsSamplingMetricsInterval=<number>", ...}

**Default:** 60
JobsSamplingMetricsInterval is a parameter of the AdvancedConfig (page 862) 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 862) directive. Its value is a time period, in seconds. This time period value elapses between the checks that CMAemon 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 862) directive. If this parameter is set to a path leading to a script, and if a new user is added using cmsh or Bright View, then the script is automatically run by CMAemon, 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 862) directive. If this parameter is set to 1, then CMAemon 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 862) directive. If this parameter is set to a path leading to a script, and if an existing user is removed using cmsh or Bright View, then the script is automatically run by CMAemon. 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 862) 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 862) directive. It sets the timeout value in seconds, for the script set by RemoveUserScript.

**AdditionalSubmitHosts directive**

**AdditionalExecHosts directive**

**Syntax:**

\[
\text{AdvancedConfig} = \{"\text{AdditionalSubmitHosts}=<\text{host1}>,<\text{host2}>,\ldots", \ldots\}
\]

or

\[
\text{AdvancedConfig} = \{"\text{AdditionalExecHosts}=<\text{host1}>,<\text{host2}>,\ldots", \ldots\}
\]

**Default: none**

The AdditionalSubmitHosts and AdditionalExecHosts directives are parameters of the AdvancedConfig (page 862) 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 SGE and UGE workload managers only, so that CMDaemon does not remove such hosts from the workload manager configuration file during a configuration file maintenance run. An example of their use is given on page 325.

**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", \ldots}

**Default: 0**

The AllowImageUpdateWithAutoMount directive is a parameter of the AdvancedConfig (page 862) directive. The values it takes decide how an auto-mounted filesystem should be dealt with during image updates (section 5.6.2) or grab image (section 12.5.2). It must be set per software image or per node.
<table>
<thead>
<tr>
<th>Value</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>If auto-mount is running, abort provisioning (default)</td>
</tr>
<tr>
<td>1</td>
<td>If auto-mount is running, warn but continue</td>
</tr>
<tr>
<td>2</td>
<td>Do not check auto-mount status. This saves a little time, but it risks data loss, unless the automounted filesystem has been added to <code>excludelistupdate</code>. If the automounted filesystem is not added to <code>excludelistupdate</code>, then if the automounted filesystem happens to be unavailable at the time that an imageupdate is issued, then the rsync process can end up deleting the automounted filesystem contents during the rsync, because it assumes that content should not be there.</td>
</tr>
<tr>
<td>3</td>
<td>Pretend auto-mount is running. This prevents an image update</td>
</tr>
</tbody>
</table>

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 247.

**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 862) 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 854) 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 862) 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@bright92 ~]# 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: none

The SSLServerMethod directive is a parameter of the AdvancedConfig (page 862) directive, as well as the GlobalConfig (page 863) 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, all TLS versions are supported. There are however only TLS 1.3 ciphers configured for the SSL server of CMDaemon, and no TLS 1.2 ciphers configured. This is due to the greater speed and security of TLS 1.3.

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". ... }

By default in this case, the SSL server of CMDaemon will fail to complete negotiation, because by default no TLS 1.2 ciphers are configured.

The SSLServerMethod directive is implied by the TLS masking syntax. The TLS masking syntax allows TLS negotiation method 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 nodes with the extra settings sometimes run extra processes for no reason. Running the directive as an AdvancedConfig is therefore suboptimal for most reasonable use cases.

There may still be some SSL clients that require a TLS 1.2 cipher and have no TLS 1.3 cipher negotiation ability. Earlier versions of CMDaemon, released prior to the NVIDIA Bright Cluster Manager subversion releases 9.1-3, 9.0-13, and 8.2-24, made TLS 1.2 ciphers available as a fallback option, so that
negotiation succeeded for SSL clients that required a TLS 1.2 cipher and had no TLS 1.3 cipher negotiation ability.

Reverting to the TLS 1.2 fallback cipher availability of earlier versions of CMDaemon is possible in the current CMDaemon by setting:

```
AdvancedConfig = { "TLS 1.1=0", "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

```
GlobalConfig = {"JobInformationDisabled=0|1", ...}
```

**Default:** 

```
JobInformationDisabled=0
```

The `JobInformationDisabled` directive is a parameter of the `AdvancedConfig` (page 862) directive, as well as the `GlobalConfig` (page 863) directive. If set to 1 it disables job-based monitoring (Chapter 14). 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 862) 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 × 3600 × 28 seconds).

### JobInformationChargeBackKeepDuration directive

**Syntax:**

```
AdvancedConfig = {"JobInformationChargeBackKeepDuration=<number>" , ...}
```

**Default:**

```
JobInformationChargeBackKeepDuration=158112000
```

The `JobInformationChargeBackKeepDuration` directive is a parameter of the `AdvancedConfig` (page 862) 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 × 3600 × 366 × 5 seconds).
JobInformationKeepCount directive

Syntax: AdvancedConfig = {"JobInformationKeepCount=<number>", ...}
Default: JobInformationKeepCount=8192

The JobInformationKeepCount directive is a parameter of the AdvancedConfig (page 862) 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 862) 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 862) 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 862) 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 862) 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 862) 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 862) 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 862) 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 response, 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 862) 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 Bright 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 Bright 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 862) directive.

By default there is a software limit of 500 measurables per data producer. If this limit is exceeded, then the CMDaemon monitoring info logs show complaints about “too many measurables”.

In that case, if the cluster hardware is not too slow, and if the measurables produced are not some kind of hardware garbage data values, then increasing the value of this directive should allow more measurables to be dealt with.
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:

```xml
<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:

```
[root@bright92 ~]# xmllint --noout --schema /cm/local/apps/cmd/etc/htdocs/xsd/disks.xsd /cm/local/apps/cmd/etc/htdocs/disk-setup/slave-diskless.xml validates
[root@bright92 ~]#
```
XML schema for partitioning

<?xml version='1.0'?>

<!--
# Copyright (c) 2004-2017 Bright Computing, Inc. All Rights Reserved.
#
# This software is the confidential and proprietary information of
# Bright Computing, Inc. ("Confidential Information"). You shall not
# disclose such Confidential Information and shall use it only in
# accordance with the terms of the license agreement you entered into
# with Bright Computing, Inc.

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:element>

</xs:schema>
<xs:element 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='8|16K|32K|64K|128K|256K|512K|1M|2M|4M|8M|16M|32M|64M|128M|256M|512M|1G|2G|4G|8G|16G|32G|64G|128G|256G|512G|1T|2T|4T|8T|16T|32T|64T|128T|256T|512T|1P|2P|4P|8P|16P|32P|64P|128P|256P|512P|1Z|2Z|4Z|8Z|16Z|32Z|64Z|128Z|256Z'/>
  </xs:restriction>
</xs:simpleType>

<xs:simpleType name="size">
</xs:simpleType>
<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]+'/>  
</xs:restriction>
</xs:simpleType>

<xs:complexType name='blockdev'>
  <xs:complexContent>
    <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:complexContent>
</xs:complexType>

<xs:complexType name='device'>
  <xs:complexContent>
    <xs:extension base='blockdev'>
      <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:complexContent>
</xs:complexType>
<xs:simpleType>
  <xs:restriction base="xs:string">
    <xs:pattern value="gpt|msdos"/>
  </xs:restriction>
</xs:simpleType>

<xs:element name="partitionTable" minOccurs="0" maxOccurs="1">
  <xs:complexType>
    <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" minOccurs="0" maxOccurs="1"/>
    </xs:sequence>
    <xs:attribute name="origin" type="xs:string"/>
  </xs:complexType>
</xs:element>

<xs:element name="partition" type="partition" minOccurs="1" maxOccurs="unbounded">
  <xs:simpleType>
    <xs:restriction base="xs:string">
      <xs:minInclusive value="0"/>
      <xs:maxInclusive value="32767"/>
    </xs:restriction>
  </xs:simpleType>
</xs:element>

<xs:element name="encryption" type="encryption" minOccurs="0" maxOccurs="1" mixed="true">
  <xs:complexType>
    <xs:sequence>
      <xs:element name="type" minOccurs="0" maxOccurs="1"/>
    </xs:sequence>
    <xs:attribute name="id" type="xs:string" use="required"/>
    <xs:attribute name="partitiontype" type="xs:string"/>
  </xs:complexType>
</xs:element>

<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" minOccurs="0" maxOccurs="1"/>
    <xs:group ref="filesystem" minOccurs="0" maxOccurs="1"/>
    <xs:element name="priority" minOccurs="0" maxOccurs="1"/>
    <xs:complexType>
      <xs:sequence>
        <xs:element name="priority" type="xs:integer">
          <xs:restriction base="xs:integer">
            <xs:minInclusive value="0"/>
            <xs:maxInclusive value="32767"/>
          </xs:restriction>
        </xs:element>
      </xs:sequence>
      <xs:attribute name="id" type="xs:string" use="required"/>
      <xs:attribute name="partitiontype" type="xs:string"/>
    </xs:complexType>
  </xs:sequence>
  <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:sequence>
  </xs:complexType>
</xs:element>
<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' minOccurs='0' maxOccurs='1'/>
    <xs:element name='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:sequence>
  <xs:attribute name='thinpool'>
    <xs:simpleType>
      <xs:restriction base="xs:string">
        <xs:pattern value="1"/>
      </xs:restriction>
    </xs:simpleType>
  </xs:attribute>
</xs:complexType>
Examples Of Element Types In XML Schema

<table>
<thead>
<tr>
<th>Name Of Type</th>
<th>Example Values</th>
</tr>
</thead>
<tbody>
<tr>
<td>size</td>
<td>10G, 128M, 1T, 2.5T, 1/3, 33.333%, auto, max</td>
</tr>
<tr>
<td>device</td>
<td>/dev/sda, /dev/hda, /dev/cciss/c0d0</td>
</tr>
<tr>
<td>partition</td>
<td>linux, linux raid, linux swap, unspecified</td>
</tr>
<tr>
<td>filesystem</td>
<td>ext2, ext3, ext4, xfs</td>
</tr>
</tbody>
</table>

D.2 Structure Of Hardware RAID Definition—The Hardware RAID XML Schema File

Hardware RAID 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 here, while schema examples are listed in section D.4.1.

<?xml version="1.0" encoding="ISO-8859-1" ?>

<!--
# Copyright (c) 2004-2014 Bright Computing, Inc. All Rights Reserved.
# This software is the confidential and proprietary information of
# Bright Computing, Inc. ("Confidential Information"). You shall not
# disclose such Confidential Information and shall use it only in
# accordance with the terms of the license agreement you entered into
# with Bright Computing, Inc.
--

<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 name="subVolumes">
  <xs:sequence>
    <xs:element name="subVolume" type="subVolume" minOccurs='1' maxOccurs='unbounded'/>
  </xs:sequence>
</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>
D.2 Structure Of Hardware RAID Definition—The Hardware RAID XML Schema File

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.

xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"

<xsd:schema xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"

<xsd:simpleType name="raidLevel">
    <xsd:restriction base="xs:nonNegativeInteger">
        <xsd:pattern value="0|1|5|10|50"/>
    </xsd:restriction>
</xsd:simpleType>

<xsd:simpleType name="volumeSize">
    <xsd:restriction base="xs:string">
        <xsd:pattern value="[0-9]{1,5}[MGT]|max"/>
    </xsd:restriction>
</xsd:simpleType>

<xsd:simpleType name="stripeSize">
    <xsd:restriction base="xs:string">
        <xsd:pattern value="8K|16K|32K|64K|128K|256K|512K|1024K"/>
    </xsd:restriction>
</xsd:simpleType>

<xsd:simpleType name="cachePolicy">
    <xsd:restriction base="xs:string">
        <xsd:pattern value="Cached|Direct"/>
    </xsd:restriction>
</xsd:simpleType>

<xsd:simpleType name="readPolicy">
    <xsd:restriction base="xs:string">
        <xsd:pattern value="NORA|RA|ADRA"/>
    </xsd:restriction>
</xsd:simpleType>

<xsd:simpleType name="writePolicy">
    <xsd:restriction base="xs:string">
        <xsd:pattern value="WT|WB"/>
    </xsd:restriction>
</xsd: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: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:choice>
    </xs:sequence>
  </xs:complexType>
</xs:element>
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
<?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>
  </device>
</diskSetup>
```
<!-- the following for paravirtual rhel6: -->
<blockdev mode="cloud">/dev/xvdf</blockdev>

<!-- the following for nvm 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> 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@bright92 ~]# cmsh
[bright92]% category use default
[bright92->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>
  </raidArray>
```
<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>
  </device>
</span>
D.5 Example: Software RAID

The following example shows a simple software RAID setup:

Example

```xml
<?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)
890 Disk Partitioning And RAID Configuration

- 6 (striping with distributed double parity)

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, RAID 10 configurations.

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.

D.6 Example: Software RAID With Swap

The <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
<?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
<?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/wdb</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
<?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>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>
```
D.9 Example: Diskless

This example shows a node configured for diskless operation:

Example

```xml
<?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
<?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 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
<?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
<?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>
  </device>
</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
<?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>max</size>
      <type>linux raid</type>
    </partition>
    ...
  </device>
</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
<?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="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>
```
<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>612M</size>
    <type>linux raid</type>
  </partition>

  <partition id="b2">
<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>

<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"/>
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
<?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>6G</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>
</diskSetup>
```
<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>6G</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>
<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>
</device>
<partition id="c4">
  <size>30G</size>
  <type>linux raid</type>
</partition>
<partition id="c5">
  <size>max</size>
  <type>linux raid</type>
</partition>
<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:

```xml
<device>
  <blockdev>/dev/vdb</blockdev>
</device>
```

<!-- One swap partition and the rest of the filesystem on 1 partition -->

```xml
<diskSetup xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance">
  <device>
    <blockdev>/dev/vdb</blockdev>
  </device>
</diskSetup>
```
<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.

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
To confirm the approval, the `installerinteractions` command can be run with the `--confirm` option and the `-w|--write` option:

**Example**

```
[bright92->device[node001]]% installerinteractions -w --confirm
[bright92->device[node001]]% installerinteractions
```

<table>
<thead>
<tr>
<th>Hostname</th>
<th>Category</th>
<th>Type</th>
<th>Timestamp</th>
<th>Action</th>
<th>Status</th>
</tr>
</thead>
<tbody>
<tr>
<td>node001</td>
<td>default</td>
<td>disk encryption passphrase</td>
<td>Wed Jun 9 12:35:22 2021</td>
<td>Waiting for approval</td>
<td>confirmed</td>
</tr>
</tbody>
</table>

```
[... after some time...]
```

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**

```
[bright92->device[node001]]% installerinteractions -w --confirm --for 5m
```

```
[bright92->device[node001]]% installerinteractions
```

<table>
<thead>
<tr>
<th>Hostname</th>
<th>Category</th>
<th>Type</th>
<th>Timestamp</th>
<th>Action</th>
<th>Status</th>
</tr>
</thead>
<tbody>
<tr>
<td>node001</td>
<td>default</td>
<td>disk encryption passphrase</td>
<td>Thu Jun 10 16:53:46 2021</td>
<td>Future action</td>
<td>confirmed</td>
</tr>
</tbody>
</table>

Possible units that can be used to specify the time period are:

<table>
<thead>
<tr>
<th>Unit</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>s</td>
<td>seconds</td>
</tr>
<tr>
<td>m</td>
<td>minutes</td>
</tr>
<tr>
<td>h</td>
<td>hours</td>
</tr>
<tr>
<td>d</td>
<td>days</td>
</tr>
<tr>
<td>w</td>
<td>weeks</td>
</tr>
</tbody>
</table>
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

[bright92->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] bright92: node001 [ INSTALLING ] (waiting for disk encryption passphrase)
[bright92->device[node001]]% installerinteractions --w -p mysecret --confirm --force

<table>
<thead>
<tr>
<th>Hostname</th>
<th>Category</th>
<th>Type</th>
<th>Timestamp</th>
<th>Action</th>
<th>Status</th>
</tr>
</thead>
<tbody>
<tr>
<td>node001</td>
<td>default</td>
<td>disk encryption passphrase</td>
<td>Fri Jun 11 23:08:36 2021</td>
<td>Waiting for approval pending</td>
<td></td>
</tr>
</tbody>
</table>

...time passes...

[bright92->device[node001]]% installerinteractions
No remote interactions pending.

[bright92->device[node001]]%
[bright92->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:

    [bright92->device[node001]]% installerinteractions --w --reset --confirm
Example initialize And finalize Scripts

The node-installer executes any initialize and finalize scripts at particular stages of its 13-step run during node-provisioning (section 5.4). They are sometimes useful for troubleshooting or workarounds during those stages. The scripts are stored in the CMDaemon database, rather than in the filesystem as plain text files, because they run before the node’s init process takes over and establishes the final filesystem.

Default initialize and finalize scripts are provided with the default category:

```
[bright92->category[default]]% show | grep ize
Initialize script <1.46KiB>
Finalize script <3.4KiB>
```

E.1 When Are They Used?

The initialize and finalize scripts are sometimes used as an alternative configuration option out of a choice of other possible options (section 3.16.1). 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 Bright View And cmsh

The initialize and finalize scripts are accessible for viewing and editing:

- In Bright 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@bright92 ~]# cmsh
[bright92% category use default
[bright92->category[default]]% show | grep script
Parameter Value
-----------------------------------------------
Finalize script <1367 bytes>
Initialize script <0 bytes>
[bright92->category[default]]% set initializescript

Example

[bright92% device use node001
[bright92->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

[bright92->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

<table>
<thead>
<tr>
<th>Variable</th>
<th>Example Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>CMD_ACTIVE_MASTER_IP</td>
<td>10.141.255.254</td>
</tr>
<tr>
<td>CMD_CATEGORY</td>
<td>default</td>
</tr>
<tr>
<td>CMD_CHASSIS</td>
<td>chassis01</td>
</tr>
<tr>
<td>CMD_CHASSIS_IP</td>
<td>10.141.1.1</td>
</tr>
<tr>
<td>CMD_CHASSIS_PASSWORD</td>
<td>ADMIN</td>
</tr>
<tr>
<td>CMD_CHASSIS_SLOT</td>
<td>1</td>
</tr>
<tr>
<td>CMD_CHASSIS_USERNAME</td>
<td>ADMIN</td>
</tr>
<tr>
<td>CMD_CLUSTERNAME</td>
<td>Bright 9.2 Cluster</td>
</tr>
<tr>
<td>CMD_DEVICE_HEIGHT</td>
<td>1</td>
</tr>
<tr>
<td>CMD_DEVICE_POSITION</td>
<td>10</td>
</tr>
<tr>
<td>CMD_DEVICE_TYPE</td>
<td>SlaveNode</td>
</tr>
<tr>
<td>CMD ETHERNETSWITCH</td>
<td>switch01:1</td>
</tr>
<tr>
<td>CMD_FSEXPORT__SLASH_cm_SLASH_node-installer_ALLOWWRITE</td>
<td>no</td>
</tr>
<tr>
<td>CMD_FSEXPORT__SLASH_cm_SLASH_node-installer_HOSTS</td>
<td>10.141.0.0/16</td>
</tr>
<tr>
<td>CMD_FSEXPORT__SLASH_cm_SLASH_node-installer_PATH</td>
<td>/cm/node-installer</td>
</tr>
<tr>
<td>CMD_FSEXPORTS</td>
<td>_SLASH_cm_SLASH_node-installer</td>
</tr>
<tr>
<td>CMD_FSMOUNT__SLASH_cm_SLASH_shared_DEVICE</td>
<td>master:/cm/shared</td>
</tr>
<tr>
<td>CMD_FSMOUNT__SLASH_cm_SLASH_shared_FILESYSTEM</td>
<td>nfs</td>
</tr>
<tr>
<td>CMD_FSMOUNT__SLASH_cm_SLASH_shared_MOUNTPOINT</td>
<td>/cm/shared</td>
</tr>
<tr>
<td>CMD_FSMOUNT__SLASH_cm_SLASH_shared_OPTIONS</td>
<td>rsize=32768,wsize=32768,</td>
</tr>
<tr>
<td></td>
<td>hard, intr, async</td>
</tr>
<tr>
<td>CMD_FSMOUNT__SLASH_dev_SLASH_pts_DEVICE</td>
<td>none</td>
</tr>
<tr>
<td>CMD_FSMOUNT__SLASH_dev_SLASH_pts_FILESYSTEM</td>
<td>devpts</td>
</tr>
<tr>
<td>CMD_FSMOUNT__SLASH_dev_SLASH_pts_MOUNTPOINT</td>
<td>/dev/pts</td>
</tr>
<tr>
<td>CMD_FSMOUNT__SLASH_dev_SLASH_pts_OPTIONS</td>
<td>gid=5, mode=620</td>
</tr>
<tr>
<td>CMD_FSMOUNT__SLASH_dev_SLASH_shm_DEVICE</td>
<td>none</td>
</tr>
<tr>
<td>CMD_FSMOUNT__SLASH_dev_SLASH_shm_FILESYSTEM</td>
<td>tmpfs</td>
</tr>
<tr>
<td>CMD_FSMOUNT__SLASH_dev_SLASH_shm_MOUNTPOINT</td>
<td>/dev/shm</td>
</tr>
<tr>
<td>CMD_FSMOUNT__SLASH_dev_SLASH_shm_OPTIONS</td>
<td>defaults</td>
</tr>
<tr>
<td>CMD_FSMOUNT__SLASH_home_DEVICE</td>
<td>master:/home</td>
</tr>
<tr>
<td>CMD_FSMOUNT__SLASH_home_FILESYSTEM</td>
<td>nfs</td>
</tr>
<tr>
<td>CMD_FSMOUNT__SLASH_home_MOUNTPOINT</td>
<td>home</td>
</tr>
<tr>
<td>CMD_FSMOUNT__SLASH_home_OPTIONS</td>
<td>rsize=32768,wsize=32768,</td>
</tr>
<tr>
<td></td>
<td>hard, intr, async</td>
</tr>
<tr>
<td>CMD_FSMOUNT__SLASH_proc_DEVICE</td>
<td>none</td>
</tr>
<tr>
<td>CMD_FSMOUNT__SLASH_proc_FILESYSTEM</td>
<td>proc</td>
</tr>
</tbody>
</table>

...continues
## Table E: Environment Variables For The initialize And Finalize Scripts...continued

<table>
<thead>
<tr>
<th>Variable</th>
<th>Example Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>CMD_FSMOUNT__SLASHprocs_MOUNTPOINT</td>
<td>/proc</td>
</tr>
<tr>
<td>CMD_FSMOUNT__SLASHprocs_OPTIONS</td>
<td>defaults,nosuid</td>
</tr>
<tr>
<td>CMD_FSMOUNT__SLASHsys_DEVICE</td>
<td>none</td>
</tr>
<tr>
<td>CMD_FSMOUNT__SLASHsys_FILESYSTEM</td>
<td>sysfs</td>
</tr>
<tr>
<td>CMD_FSMOUNT__SLASHsys_MOUNTPOINT</td>
<td>/sys</td>
</tr>
<tr>
<td>CMD_FSMOUNT__SLASHsys_OPTIONS</td>
<td>defaults</td>
</tr>
<tr>
<td>CMD_FSMOUNTS */SLASH_dev_Slash_pts</td>
<td>_SLASH_dev_Slash_pts</td>
</tr>
<tr>
<td>CMD_GATEWAY</td>
<td>10.141.255.254</td>
</tr>
<tr>
<td>CMD_HOSTNAME</td>
<td>node001</td>
</tr>
<tr>
<td>CMD_INSTALLMODE</td>
<td>AUTO</td>
</tr>
<tr>
<td>CMD_INTERFACE_eth0_IP</td>
<td>10.141.0.1</td>
</tr>
<tr>
<td>CMD_INTERFACE_eth0_MTU</td>
<td>1500</td>
</tr>
<tr>
<td>CMD_INTERFACE_eth0_NETMASK</td>
<td>255.255.0.0</td>
</tr>
<tr>
<td>CMD_INTERFACE_eth0_TYPE</td>
<td>physical</td>
</tr>
<tr>
<td>CMD_INTERFACES</td>
<td>eth0 eth1 eth2 ipmi0</td>
</tr>
<tr>
<td>CMD_IP</td>
<td>10.141.0.1</td>
</tr>
<tr>
<td>CMD_MAC</td>
<td>00:00:00:00:00:00:00:01</td>
</tr>
<tr>
<td>CMD_PARTITION</td>
<td>base</td>
</tr>
<tr>
<td>CMD_PASSIVE_MASTER_IP</td>
<td>10.141.255.253</td>
</tr>
<tr>
<td>CMD_PDUS</td>
<td>custom</td>
</tr>
<tr>
<td>CMD_RACK</td>
<td>rack01</td>
</tr>
<tr>
<td>CMD_RACK_HEIGHT</td>
<td>42</td>
</tr>
<tr>
<td>CMD_RACK_Room</td>
<td>serverroom</td>
</tr>
<tr>
<td>CMD_ROLES</td>
<td>sgeclient storage</td>
</tr>
<tr>
<td>CMD_SHARED_MASTER_IP</td>
<td>10.141.255.252</td>
</tr>
<tr>
<td>CMD_SOFTWAREIMAGE_PATH</td>
<td>/cm/images/default-image</td>
</tr>
<tr>
<td>CMD_SOFTWAREIMAGE</td>
<td>default-image</td>
</tr>
<tr>
<td>CMD_TAG</td>
<td>0000000000a000</td>
</tr>
<tr>
<td>CMD_USERDEFINED1</td>
<td>var1</td>
</tr>
<tr>
<td>CMD_USERDEFINED2</td>
<td>var2</td>
</tr>
</tbody>
</table>

* The value for this variable is a string with spaces, not an array. Eg:

CMD_FSMOUNTS="_SLASH_dev_Slash_pts _SLASH_procs _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 finalize script
#!/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:

<table>
<thead>
<tr>
<th>Description</th>
<th>Naming Convention</th>
</tr>
</thead>
<tbody>
<tr>
<td>volume</td>
<td>CMD_FSMOUNT_&lt;x&gt;_DEVICE</td>
</tr>
<tr>
<td>mount point</td>
<td>CMD_FSMOUNT_&lt;x&gt;_MOUNTPOINT</td>
</tr>
<tr>
<td>filesystem type</td>
<td>CMD_FSMOUNT_&lt;x&gt;_FILESYSTEM</td>
</tr>
<tr>
<td>mount point options</td>
<td>CMD_FSMOUNT_&lt;x&gt;_OPTIONS</td>
</tr>
</tbody>
</table>
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_<y>_SLASH_cm_SLASH_shared_DEVICE.

A similar naming convention is applicable to the names of the environment variables for the export entries:

<table>
<thead>
<tr>
<th>Description</th>
<th>Naming Convention</th>
</tr>
</thead>
<tbody>
<tr>
<td>exported system writable?</td>
<td>CMD_FSEXPORT_&lt;y&gt;_ALLOWWRITE</td>
</tr>
<tr>
<td>allowed hosts or networks</td>
<td>CMD_FSEXPORT_&lt;y&gt;_HOSTS</td>
</tr>
<tr>
<td>path on exporter</td>
<td>CMD_FSMOUNT_&lt;y&gt;_PATH</td>
</tr>
</tbody>
</table>

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 Bright 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**

```bash
#!/bin/bash
cp /etc/myapp.conf.overwrite /localdisk/etc/myapp.conf
```

This technique is used in a finalize script example in section 3.16.4, 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 657.

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 3.16.4 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**

```bash
#!/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:

```bash
#!/bin/bash
perl -p -i -e 's/\$\{\([-\)]+\}\$/\$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.
Workload Managers Quick Reference

F.1 Slurm

Slurm is a GPL-licensed workload management system and developed largely at Lawrence Livermore National Laboratory. The name was originally an acronym for Simple Linux Utility for Resource Management, but the acronym is deprecated because it no longer does justice to the advanced capabilities of Slurm.

The Slurm service and outputs are normally handled using the Bright View or cmsh front end tools for CMDaemon (section 7.4).

From the command line, direct Slurm commands that may sometimes come in useful include the following:

- `sacct`: used to report job or job step accounting information about active or completed jobs.

  Example

  ```bash
  # sacct -j 43 -o jobid,AllocCPUs,NCPUS,NNodes,NTasks,ReqCPUs
  JobID  AllocCPUs  NCPUS  NNodes  NTasks  ReqCPUs
  --------  ---------  -------  --------  -----  --------
  43      1         1       1        1      1
  ```

- `salloc`: used to allocate resources for a job in real time. Typically this is used to allocate resources and spawn a shell. The shell is then used to execute srun commands to launch parallel tasks.

- `sattach`: used to attach standard input, output, and error plus signal capabilities to a currently running job or job step. One can attach to and detach from jobs multiple times.

- `sbatch`: used to submit a job script for later execution. The script typically contains one or more srun commands to launch parallel tasks.

- `sbcast`: used to transfer a file from local disk to local disk on the nodes allocated to a job. This can be used to effectively use diskless compute nodes or provide improved performance relative to a shared filesystem.

- `scancel`: used to cancel a pending or running job or job step. It can also be used to send an arbitrary signal to all processes associated with a running job or job step.

- `scontrol`: the administrative tool used to view and/or modify Slurm state. Note that many scontrol commands can only be executed as user root.

  Example
If a node, for example \texttt{node001}, is stuck in a \texttt{CG} state ("completing"), and rebooting it is not feasible, then the following may clear it in some cases:

**Example**

\[
\texttt{[fred@bright92 ~] } \$ \texttt{scontrol update nodename=node001 state=down reason=hung} \\
\texttt{[fred@bright92 ~] } \$ \texttt{scontrol update nodename=node001 state=resume}
\]

- \texttt{sinfo}: reports the state of partitions and nodes managed by Slurm. It has a wide variety of filtering, sorting, and formatting options.

**Example**

\[
\texttt{bright92:~ # sinfo -o "%9P %.5a %.10l %.6D %.6t %C %N"}
\]

- \texttt{srun}: used to submit a job for execution or initiate job steps in real time. \texttt{srun} has a wide variety of options to specify resource requirements, including: minimum and maximum node count, processor count, specific nodes to use or not use, and specific node characteristics (so much memory, disk space, certain required features, etc.). A job can contain multiple job steps executing sequentially or in parallel on independent or shared nodes within the job’s node allocation.

- \texttt{sview}: a graphical user interface to get and update state information for jobs, partitions, and nodes managed by Slurm.

There are man pages for these commands. Full documentation on Slurm is available online at: \url{http://slurm.schedmd.com/documentation.html}. 
F.2 Univa Grid Engine

Univa Grid Engine (UGE) is a workload management system that was originally derived from Sun Grid Engine (SGE).

UGE services should be handled using CMDaemon, as explained in section 7.4.

Managing queues and jobs can be done via the cmsh or Bright View front ends, but can also be carried out via the UGE tools.

- The `sge_qmaster` daemon on the head node can be started, stopped, or viewed using `service sge_qmaster start | stop | status`, or alternatively handled at a lower, more fine-grained, level using the `-k` (kill), `-s` (show) and `-a` (activate) options of `qconf`. Further details on these options can be found in the `qconf` help text, or `man qconf(1)`. For example, help on the `-k` options can be found by running:
  ```
  qconf -help | grep ' '-k'
  ```
  or running:
  ```
  man qconf(1) and scrolling down to the interesting part by searching for `shutdown`).
  ```
- The `sge_execd` execution daemon running on each compute node accepts, manages, and returns the results of the jobs on the compute nodes. The daemon can be managed via `service sge_execd start | stop | status`, or alternatively (de)registered from `qmaster` via `qconf` options.
- To see why a queue is in an error state, `qstat -explain E` shows the reason. Further details on these options can be found in the `qconf` help text, or `man qstat(1)`. Queues in an error state can be cleared with a `qmod -cq <queue name>`.

UGE can be configured and managed generally with the command line utility `qconf`, which is what most administrators become familiar with. UGE also provides a GUI alternative, `qmon`.

UGE commands are listed below. The details of these are in the `man` page of the command and the UGE documentation.

- `qalter`: modify existing batch jobs
- `qacct`: show usage information from accounting data
- `qconf`: configure UGE
- `qdel`: delete batch jobs
- `qhold`: place hold on batch jobs
- `qhost`: display compute node queues, states, jobs
- `qlogin`: start login-based interactive session with a node
- `qmake`: distributed, parallel make utility
- `qmod`: suspend/enable queues and jobs
- `qmon`: configure UGE with an X11 GUI interface
- `qping`: check `sge_qmaster` and `sge_execd` status
- `qquot`: list resource quotas
- `qresub`: create new jobs by copying existing jobs
- `qrdel`: cancel advance reservations
• **qrls**: release batch jobs from a held state
• **qrsh**: start rsh-based interactive session with node
• **qrstat**: show status of advance reservations
• **qrsub**: submit advanced reservation
• **qselect**: select queues based on argument values
• **qsh**: start sh interactive session with a node
• **qstat**: show status of batch jobs and queues
• **qsub**: submit new jobs (related: qalter, qresub)

### F.3 PBS Pro

The following commands can be used in PBS Pro to view queues, jobs, and server status:

- `qstat`: query queue status
- `qstat -a`: show only queued or running jobs for a destination, or all states for a job ID
- `qstat -r`: show only running or suspended jobs for a destination, or all states for a job ID
- `qstat -q`: show queue status for destinations
- `qstat -rn`: only running or suspended jobs, with list of allocated nodes (exec_host string)
- `qstat -i`: information on queued, held, waiting jobs is given for specified destination. Information about the job is given if a job ID is specified, regardless of the job status
- `qstat -B`: display server status for the specified servers
- `qstat -u <username>`: show jobs for a user for the specified destination. Status information for the job is displayed for a specified job ID.

Other useful commands are:

- `tracejob <job id>`: show what happened today to <job id>
- `tracejob -n <number> <job id>`: search last <number> days for <job id>

- `qmgr`: administrator interface to batch system (man 8B qmgr for more details)
- `qterm`: terminates PBS server (but the cluster manager starts pbs_server again)

- `pbsnodes <node>`: query status of compute node
- `pbsnodes -a`: query status of all compute nodes

The commands of PBS Pro are documented in the man pages, and also in the extensive documentation available at [https://www.altair.com/pbs-works-documentation/](https://www.altair.com/pbs-works-documentation/)
Metrics, Health Checks, Enummetrics, And Actions

This appendix describes the metrics (section G.1), health checks (section G.2), enummetrics (section 13.2.2), and actions (section G.4), along with their parameters, in a newly-installed cluster. Metrics, health checks, enummetrics, and actions can each be standalone scripts, or they can be built-ins. Standalone scripts can be those supplied with the system, or they can be custom scripts built by the administrator. Scripts often require environment variables (as described in section 3.3.1 of the Developer Manual). On success scripts must exit with a status of 0, as is the normal practice.

G.1 Metrics And Their Parameters

A list of metric names can be viewed, for example, for the head node, using cmsh as follows (section 13.5.3):

[bright92 ~]# cmsh -c "monitoring measurable; list metric"

The metrics are listed in this section are classed into 9 kinds:

1. regular metrics (section G.1.1)
2. NFS metrics (section G.1.2)
3. InfiniBand metrics (section G.1.3)
4. monitoring system metrics (section G.1.4)
5. GPU metrics (section G.1.5)
6. Job metrics (section G.1.6)
7. IPMI metrics (section G.1.7)
8. SMART metrics (section G.1.8)
9. Prometheus metrics (section G.1.9)
## G.1.1 Regular Metrics

<table>
<thead>
<tr>
<th>Metric</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>AlertLevel</td>
<td>Indicates the healthiness of a device based on severity of events (section 13.2.8). The lower it is, the better. There are 3 parameters it can take:</td>
</tr>
<tr>
<td></td>
<td>• <strong>count</strong>: the number of active triggers</td>
</tr>
<tr>
<td></td>
<td>• <strong>maximum</strong>: the maximum alert level of all active triggers</td>
</tr>
<tr>
<td></td>
<td>• <strong>sum</strong>: the average alert level of all active triggers</td>
</tr>
<tr>
<td>AvgJobDuration</td>
<td>Average job duration of current jobs</td>
</tr>
<tr>
<td>AvgJobStartDelay</td>
<td>Average job start delay of current jobs</td>
</tr>
<tr>
<td>BlockedProcesses</td>
<td>Blocked processes waiting for I/O</td>
</tr>
<tr>
<td>bright_licensed_nodes</td>
<td>Nodes licensed by the cluster manager</td>
</tr>
<tr>
<td>bright_pay_by_use_nodes</td>
<td>Nodes that are pay-by-use</td>
</tr>
<tr>
<td>bright_used_nodes</td>
<td>Nodes used by the cluster manager</td>
</tr>
<tr>
<td>BufferMemory</td>
<td>System memory used for buffering</td>
</tr>
<tr>
<td>BytesRecv*‡</td>
<td>Bytes/s received</td>
</tr>
<tr>
<td>BytesSent*‡</td>
<td>Bytes/s sent</td>
</tr>
<tr>
<td>CPUGuest*</td>
<td>CPU time spent in guest mode.</td>
</tr>
<tr>
<td>CPUIdle*</td>
<td>CPU time spent in idle mode.</td>
</tr>
<tr>
<td>CPUIrq*</td>
<td>CPU time spent in servicing IRQ.</td>
</tr>
<tr>
<td>CPUNice*</td>
<td>CPU time spent in nice mode.</td>
</tr>
<tr>
<td>CPUSoftIrq*</td>
<td>CPU time spent in servicing soft IRQ.</td>
</tr>
<tr>
<td>CPUSteal*</td>
<td>CPU time spent in steal mode.</td>
</tr>
<tr>
<td>CPUSystem*</td>
<td>CPU time spent in system mode.</td>
</tr>
<tr>
<td>CPUUser*</td>
<td>CPU time spent in user mode.</td>
</tr>
<tr>
<td>CPUNeat*</td>
<td>CPU time spent in I/O wait mode.</td>
</tr>
<tr>
<td>CacheMemory</td>
<td>System memory used for caching.</td>
</tr>
<tr>
<td>CompletedJobs</td>
<td>Successfully completed jobs</td>
</tr>
<tr>
<td>Cores</td>
<td>Number of cores for a node</td>
</tr>
<tr>
<td>CoresInQueue</td>
<td>Cores in queue</td>
</tr>
<tr>
<td>CoresTotal</td>
<td>Total number of known cores for all nodes</td>
</tr>
<tr>
<td>CoresUPInQueue</td>
<td>Active cores in queue</td>
</tr>
<tr>
<td>CoresUp</td>
<td>Number of cores for all nodes marked as UP</td>
</tr>
<tr>
<td>CtxtSwitches*</td>
<td>Context switches/s</td>
</tr>
<tr>
<td>DevicesClosed</td>
<td>Number of devices not marked as UP or DOWN</td>
</tr>
<tr>
<td>DevicesDown</td>
<td>Number of devices marked as DOWN</td>
</tr>
</tbody>
</table>

...continues
<table>
<thead>
<tr>
<th>Metric</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>DevicesTotal</td>
<td>Total number of devices</td>
</tr>
<tr>
<td>DevicesUp</td>
<td>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.</td>
</tr>
<tr>
<td>DropRecv&lt;sup&gt;*,†&lt;/sup&gt;</td>
<td>Packets/s received and dropped</td>
</tr>
<tr>
<td>DropSent&lt;sup&gt;*,†&lt;/sup&gt;</td>
<td>Packets/s sent and dropped</td>
</tr>
<tr>
<td>EC2SpotPrice</td>
<td>Amazon EC2 price for spot instances</td>
</tr>
<tr>
<td>EccDBitGPU&lt;sup&gt;**,†&lt;/sup&gt;</td>
<td>Total number of double bit ECC errors (file: <code>sample_gpu</code>)</td>
</tr>
<tr>
<td>EccSBitGPU&lt;sup&gt;**,†&lt;/sup&gt;</td>
<td>Total number of single bit ECC errors (file: <code>sample_gpu</code>)</td>
</tr>
<tr>
<td>ErrorsRecv&lt;sup&gt;*,†&lt;/sup&gt;</td>
<td>Packets/s received with error</td>
</tr>
<tr>
<td>ErrorsSent&lt;sup&gt;*,†&lt;/sup&gt;</td>
<td>Packets/s sent with error</td>
</tr>
<tr>
<td>EstimatedDelay</td>
<td>Estimated delay time to execute jobs</td>
</tr>
<tr>
<td>FPGAsTotal</td>
<td>Total number of known FPGAs for all nodes</td>
</tr>
<tr>
<td>FPGAsUp</td>
<td>Number of FPGAs for all nodes marked as UP</td>
</tr>
<tr>
<td>FabricTopologies</td>
<td>Number of fabric topologies</td>
</tr>
<tr>
<td>FabricTopologyHostUsage</td>
<td>Average usage of all topology hosts</td>
</tr>
<tr>
<td>FabricTopologyResourceBoxUsage</td>
<td>Average usage of all topology resource boxes</td>
</tr>
<tr>
<td>FailedJobs</td>
<td>Failed completed jobs</td>
</tr>
<tr>
<td>fan_speed</td>
<td>Speed of a fan from Redfish</td>
</tr>
<tr>
<td>Forks&lt;sup&gt;*&lt;/sup&gt;</td>
<td>Forked processes/s</td>
</tr>
<tr>
<td>FrameErrors&lt;sup&gt;*,†&lt;/sup&gt;</td>
<td>Packet framing errors/s</td>
</tr>
<tr>
<td>FreeFiles&lt;sup&gt;§&lt;/sup&gt;</td>
<td>Free file inodes on the specified mount point</td>
</tr>
<tr>
<td>FreeSpace&lt;sup&gt;§&lt;/sup&gt;</td>
<td>Free space for non-root user. Takes mount point as a parameter</td>
</tr>
<tr>
<td>GPUUnitsClosed</td>
<td>Number of GPU units not marked as UP or DOWN</td>
</tr>
<tr>
<td>GPUUnitsDown</td>
<td>Number of GPU units marked as DOWN</td>
</tr>
<tr>
<td>GPUUnitsTotal</td>
<td>Total number of GPU Units</td>
</tr>
<tr>
<td>GPUUnitsUp</td>
<td>Number of GPU units marked as UP</td>
</tr>
<tr>
<td>GPUsTotal</td>
<td>Total number of known GPUs for all nodes</td>
</tr>
<tr>
<td>GPUsUp</td>
<td>Number of GPUs for all nodes marked as UP</td>
</tr>
<tr>
<td>HardwareCorruptedMemory</td>
<td>Hardware corrupted memory detected by ECC</td>
</tr>
<tr>
<td>IOProgress&lt;sup&gt;†&lt;/sup&gt;</td>
<td>I/O operations in progress</td>
</tr>
<tr>
<td>IOTime&lt;sup&gt;*,†&lt;/sup&gt;</td>
<td>I/O operations time in milliseconds/s</td>
</tr>
<tr>
<td>Interrupts&lt;sup&gt;*&lt;/sup&gt;</td>
<td>System interrupts</td>
</tr>
<tr>
<td>IpFormDatagrams&lt;sup&gt;*&lt;/sup&gt;</td>
<td>Input IP datagrams/s to be forwarded</td>
</tr>
<tr>
<td>IpFragCreates&lt;sup&gt;*&lt;/sup&gt;</td>
<td>IP datagram fragments/s generated</td>
</tr>
<tr>
<td>IpFragFails&lt;sup&gt;*&lt;/sup&gt;</td>
<td>IP datagrams/s which needed to be fragmented but could not</td>
</tr>
<tr>
<td>IpFragOKs&lt;sup&gt;*&lt;/sup&gt;</td>
<td>IP datagrams/s successfully fragmented</td>
</tr>
<tr>
<td>IpInAddrErrors&lt;sup&gt;*&lt;/sup&gt;</td>
<td>Input datagrams/s discarded because the IP address in their header was not a valid address</td>
</tr>
</tbody>
</table>

...continues
<table>
<thead>
<tr>
<th>Metric</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>IpInDelivers</td>
<td>Input IP datagrams/s successfully delivered</td>
</tr>
<tr>
<td>IpInDiscards</td>
<td>Input IP datagrams/s discarded</td>
</tr>
<tr>
<td>IpInHdrErrors</td>
<td>Input IP datagrams/s discarded due to errors in their IP headers</td>
</tr>
<tr>
<td>IpInReceives</td>
<td>Input IP datagrams/s, including ones with errors, received from all interfaces</td>
</tr>
<tr>
<td>IpInUnknownProtos</td>
<td>Input IP datagrams/s received but discarded due to an unknown or unsupported protocol</td>
</tr>
<tr>
<td>IpOutDiscards</td>
<td>Output IP datagrams/s discarded</td>
</tr>
<tr>
<td>IpOutNoRoutes</td>
<td>Output IP datagrams/s discarded because no route could be found</td>
</tr>
<tr>
<td>IpOutRequests</td>
<td>Output IP datagrams/s supplied to IP in requests for transmission</td>
</tr>
<tr>
<td>IpReasmOKs</td>
<td>IP datagrams/s successfully re-assembled</td>
</tr>
<tr>
<td>IpReasmReqds</td>
<td>IP fragments/s received needing re-assembly</td>
</tr>
<tr>
<td>isilon_node_disk_access_latency</td>
<td>Isilon access latency for node disk</td>
</tr>
<tr>
<td>isilon_node_disk_iosched_queue</td>
<td>Isilon iosched queue for node disk</td>
</tr>
<tr>
<td>isilon_node_disk_xfer_size_in</td>
<td>Transfer size in for node disk</td>
</tr>
<tr>
<td>isilon_node_disk_xfer_size_out</td>
<td>Transfer size out for node disk</td>
</tr>
<tr>
<td>isilon_node_ip</td>
<td>IP address of isilon node</td>
</tr>
<tr>
<td>isilon_in_rate</td>
<td>Bytes written to NFS client from Isilon</td>
</tr>
<tr>
<td>isilon_out_rate</td>
<td>Bytes read from Isilon to NFS client</td>
</tr>
<tr>
<td>isilon_op_rate</td>
<td>I/O rate for NFS client to/from Isilon</td>
</tr>
<tr>
<td>JobThroughput</td>
<td>Average number of Jobs finished</td>
</tr>
<tr>
<td>JobsRunning</td>
<td>Jobs running on the node</td>
</tr>
<tr>
<td>LiteNodesClosed</td>
<td>Number of lite nodes not marked as UP or DOWN</td>
</tr>
<tr>
<td>LiteNodesDown</td>
<td>Number of lite nodes marked as DOWN</td>
</tr>
<tr>
<td>LiteNodesTotal</td>
<td>Total number of lite nodes</td>
</tr>
<tr>
<td>LiteNodesUp</td>
<td>Number of lite nodes marked as UP</td>
</tr>
<tr>
<td>LoadFifteen</td>
<td>Load average on 15 minutes</td>
</tr>
<tr>
<td>LoadFive</td>
<td>Load average on 5 minutes</td>
</tr>
<tr>
<td>LoadOne</td>
<td>Load average on 1 minute</td>
</tr>
<tr>
<td>MajorPageFaults</td>
<td>Page faults/s that require I/O</td>
</tr>
<tr>
<td>MemoryAvailable</td>
<td>Available system memory</td>
</tr>
<tr>
<td>MemoryFree</td>
<td>Free system memory</td>
</tr>
<tr>
<td>MemoryTotal</td>
<td>Total system memory</td>
</tr>
<tr>
<td>MemoryUsed</td>
<td>Used system memory</td>
</tr>
<tr>
<td>MemoryUsed ‡</td>
<td>Used system memory for process, but default for cmd</td>
</tr>
<tr>
<td>MemoryUtilization</td>
<td>Memory utilization</td>
</tr>
<tr>
<td>MergedReads †</td>
<td>Merged reads/s</td>
</tr>
<tr>
<td>MergedWrites †</td>
<td>Merged writes/s</td>
</tr>
</tbody>
</table>

...continues
### G.1 Metrics And Their Parameters

<table>
<thead>
<tr>
<th>Metric</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>NodesClosed</td>
<td>Number of nodes not marked as UP or DOWN</td>
</tr>
<tr>
<td>NodesDown</td>
<td>Number of nodes marked as DOWN</td>
</tr>
<tr>
<td>NodesInQueue</td>
<td>Number of nodes in the queue</td>
</tr>
<tr>
<td>NodesTotal</td>
<td>Total number of nodes</td>
</tr>
<tr>
<td>NodesUpInQueue</td>
<td>Active nodes in queue</td>
</tr>
<tr>
<td>NodesUp</td>
<td>Number of nodes in status UP</td>
</tr>
<tr>
<td>OccupationRate</td>
<td>Cluster occupation rate—a normalized cluster load percentage. 100% means all cores on all nodes are fully loaded. The calculation is done as follows: ( \text{LoadOne} ) on each node is mapped to a value, calibrated so that ( \text{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 ( \text{OccupationRate} ). A high value can indicate the cluster is being used optimally. However, a value that is 100% most of the time suggests the cluster may need to be expanded.</td>
</tr>
<tr>
<td>OpaHfiTxByteCount(^*,‡)</td>
<td>Intel Omni-Path HFI Transmitted data</td>
</tr>
<tr>
<td>OpaHfiTxPktsCount(^*,‡)</td>
<td>Intel Omni-Path HFI Transmitted packets</td>
</tr>
<tr>
<td>OpaHfiRxByteCount(^*,‡)</td>
<td>Intel Omni-Path HFI Received data</td>
</tr>
<tr>
<td>OpaHfiRxPktsCount(^*,‡)</td>
<td>Intel Omni-Path HFI Received packets</td>
</tr>
<tr>
<td>PacketsRecv(^*,‡)</td>
<td>Packets/s received</td>
</tr>
<tr>
<td>PacketsSent(^*,‡)</td>
<td>Packets/s sent</td>
</tr>
<tr>
<td>PageFaults(^*)</td>
<td>Page faults/s</td>
</tr>
<tr>
<td>PageIn(^*)</td>
<td>Number of bytes the system has paged in from disk</td>
</tr>
<tr>
<td>PageOut(^*)</td>
<td>Number of bytes the system has paged out to disk</td>
</tr>
<tr>
<td>PageSwapIn(^*)</td>
<td>Number of bytes the system has swapped in from disk</td>
</tr>
<tr>
<td>PageSwapOut(^*)</td>
<td>Number of bytes the system has swapped out to disk</td>
</tr>
<tr>
<td>PDUBankLoad</td>
<td>Total PDU bank load, in amps</td>
</tr>
<tr>
<td>PDULoad</td>
<td>Total PDU phase load, in amps</td>
</tr>
<tr>
<td>PDUUptime(^*)</td>
<td>PDU uptime per second. I.e. ideally=1, but in practice has jitter effects.</td>
</tr>
<tr>
<td>PhaseLoad</td>
<td>Sum of PDULoad over all power distribution units</td>
</tr>
<tr>
<td>ProcessCount</td>
<td>Total number of all processes in the OS. These are running processes (RunningProcesses) and blocked processes(BlockedProcesses).</td>
</tr>
<tr>
<td>psu_power</td>
<td>The average power output in Watts of power supply, from Redfish</td>
</tr>
<tr>
<td>QueuedJobs</td>
<td>Queued jobs</td>
</tr>
<tr>
<td>ReadOnly(^\§)</td>
<td>Indicates if the specified mount point was mounted as read-only</td>
</tr>
<tr>
<td>ReadTime(^*,†)</td>
<td>Read time in milliseconds/s</td>
</tr>
<tr>
<td>Reads(^*,†)</td>
<td>Reads/s completed successfully</td>
</tr>
<tr>
<td>RunningJobs</td>
<td>Running jobs</td>
</tr>
</tbody>
</table>

...continues
Table G.1.1: List Of Metrics...continued

<table>
<thead>
<tr>
<th>Metric</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>RunningJobsMax</td>
<td>Maximal running jobs</td>
</tr>
<tr>
<td>RunningJobsUtilization</td>
<td>Running jobs utilization</td>
</tr>
<tr>
<td>RunningProcesses</td>
<td>Running processes</td>
</tr>
<tr>
<td>SectorsRead*\†</td>
<td>Sectors/s read successfully/s</td>
</tr>
<tr>
<td>SectorsWritten*\†</td>
<td>Sectors/s written successfully</td>
</tr>
<tr>
<td>sensor_reading</td>
<td>Reading of a temperature sensor, from Redfish</td>
</tr>
<tr>
<td>SwapCached</td>
<td>Cached swap memory</td>
</tr>
<tr>
<td>SwapFree</td>
<td>Free swap memory</td>
</tr>
<tr>
<td>SwapTotal</td>
<td>Total swap memory</td>
</tr>
<tr>
<td>SwapUsed</td>
<td>Used swap memory</td>
</tr>
<tr>
<td>SwapUtilization</td>
<td>Swap memory utilization</td>
</tr>
<tr>
<td>SystemTime\¶</td>
<td>System time used by process, by default CMDaemon</td>
</tr>
<tr>
<td>TcpCurrEstab</td>
<td>TCP connections that are either ESTABLISHED or CLOSE-WAIT</td>
</tr>
<tr>
<td>TcpInErrs*</td>
<td>Input IP segments/s received in error</td>
</tr>
<tr>
<td>TcpRetransSegs*</td>
<td>Total number of IP segments/s re-transmitted</td>
</tr>
<tr>
<td>ThreadsUsed\¶</td>
<td>Threads used by process, by default CMDaemon</td>
</tr>
<tr>
<td>TotalCPUIdle</td>
<td>Cluster-wide core usage in idle tasks (sum of all CPUIdle metric percentages)</td>
</tr>
<tr>
<td>TotalCPUSystem</td>
<td>Cluster-wide core usage in system mode (sum of all CPUSystem metric percentages)</td>
</tr>
<tr>
<td>TotalCPUUser</td>
<td>Cluster-wide core usage in user mode (sum of all CPUUser metric percentages)</td>
</tr>
<tr>
<td>TotalGPUPowerUsage</td>
<td>Total GPU power usage gnu_power_usage (table G.1.5) over all nodes</td>
</tr>
<tr>
<td>TotalGPUUtilization</td>
<td>Average of gnu_utilization:average (table G.1.5) over all nodes</td>
</tr>
<tr>
<td>TotalMemory</td>
<td>Sum of MemoryTotal over all nodes</td>
</tr>
<tr>
<td>TotalMemoryFree</td>
<td>Cluster-wide total of memory free</td>
</tr>
<tr>
<td>TotalMemoryUsed</td>
<td>Cluster-wide total of memory used</td>
</tr>
<tr>
<td>TotalSwap</td>
<td>Sum of SwapTotal over all nodes</td>
</tr>
<tr>
<td>TotalSwapFree</td>
<td>Cluster-wide total swap free</td>
</tr>
<tr>
<td>TotalSwapUsed</td>
<td>Cluster-wide total swap used</td>
</tr>
<tr>
<td>TotalUser</td>
<td>Total number of known users</td>
</tr>
<tr>
<td>TotalUserLogin</td>
<td>Total number of logged in users</td>
</tr>
<tr>
<td>UdpInDatagrams*</td>
<td>Input UDP datagrams/s delivered to UDP users</td>
</tr>
<tr>
<td>UdpInErrors*</td>
<td>Input UDP datagrams/s received that could not be delivered/s for other reasons (no port excl.)</td>
</tr>
<tr>
<td>UdpNoPorts*</td>
<td>Received UDP datagrams/s for which there was no application at the destination port</td>
</tr>
<tr>
<td>UniqueUserLogin</td>
<td>Number of unique users logged in</td>
</tr>
<tr>
<td>UnmanagedNodesClosed</td>
<td>Number of unmanaged nodes not marked as UP or DOWN</td>
</tr>
</tbody>
</table>

...continues
### Table G.1.1: List Of Metrics...continued

<table>
<thead>
<tr>
<th>Metric</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>UnmanagedNodesDown</td>
<td>Number of unmanaged nodes marked as DOWN</td>
</tr>
<tr>
<td>UnmanagedNodesTotal</td>
<td>Total number of unmanaged nodes</td>
</tr>
<tr>
<td>UnmanagedNodesUp</td>
<td>Number of unmanaged nodes marked as UP</td>
</tr>
<tr>
<td>Uptime</td>
<td>System uptime per second. Ie, ideally=1, but in practice has jitter effects</td>
</tr>
<tr>
<td>UsedFiles§</td>
<td>Used file inodes on the specified mount point</td>
</tr>
<tr>
<td>UsedSpace§</td>
<td>Used space on the specified mount point</td>
</tr>
<tr>
<td>UserTime¶</td>
<td>User time used by process, by default CMDaemon</td>
</tr>
<tr>
<td>VirtualMemoryUsed¶</td>
<td>Virtual memory used by process, by default CMDaemon</td>
</tr>
<tr>
<td>WlmSlotsFree</td>
<td>The number of WLM slots in free</td>
</tr>
<tr>
<td>WlmSlotsTotal</td>
<td>The total number of WLM slots</td>
</tr>
<tr>
<td>WlmSlotsUsed</td>
<td>The number of WLM slots in use</td>
</tr>
<tr>
<td>WlmSlotsUtilization</td>
<td>The percentage of WLM slots in use</td>
</tr>
<tr>
<td>wlm_slurm_state_count</td>
<td>Number of nodes in the state for a parameter. The possible parameters are:</td>
</tr>
<tr>
<td></td>
<td>• allocated</td>
</tr>
<tr>
<td></td>
<td>• completing</td>
</tr>
<tr>
<td></td>
<td>• down</td>
</tr>
<tr>
<td></td>
<td>• drain</td>
</tr>
<tr>
<td></td>
<td>• draining</td>
</tr>
<tr>
<td></td>
<td>• fail</td>
</tr>
<tr>
<td></td>
<td>• idle</td>
</tr>
<tr>
<td></td>
<td>• maint</td>
</tr>
<tr>
<td></td>
<td>• mixed</td>
</tr>
</tbody>
</table>

...continues
<table>
<thead>
<tr>
<th>Metric</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>WriteTime(^*),(^†)</td>
<td>Write time in milliseconds/s (this is a per mille)</td>
</tr>
<tr>
<td>Writes(^*),(^†)</td>
<td>Writes/s completed successfully</td>
</tr>
</tbody>
</table>

\(^*\) Cumulative metric. I.e. the metric is derived from cumulative raw measurements taken at two different times, according to:
\[
\text{metric}_{\text{time}_2} = \frac{\text{measurement}_{\text{time}_2} - \text{measurement}_{\text{time}_1}}{\text{time}_2 - \text{time}_1}
\]

\(^\ast\) Standalone scripts, not built-ins.

\(^\dagger\) Takes block device name (sda, sdc, nvme0n1 and so on) as parameter

\(^\ddagger\) Takes interface device name (eth0, eth1, en01, docker0, ib0 and so on) as parameter

\(^\S\) Takes mount point (for example: /, or /var) as parameter

\(^\P\) Takes a process, by default cmd, as parameter

### G.1.2 NFS Metrics

The NFS metrics are all cumulative. They correspond to nfsstat output, and are shown in table G.1.2.

<table>
<thead>
<tr>
<th>NFS Metric</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>nfs_client_packet_packets</td>
<td>NFS client packets statistics: packets</td>
</tr>
<tr>
<td>nfs_client_packet_tcp</td>
<td>NFS client package statistics: TCP/IP packets</td>
</tr>
<tr>
<td>nfs_client_packet_tcpconn</td>
<td>NFS client package statistics: TCP/IP connections</td>
</tr>
<tr>
<td>nfs_client_package_udp</td>
<td>NFS client package statistics: UDP packets</td>
</tr>
<tr>
<td>nfs_client_rpc_authrefresh</td>
<td>NFS Client RPC statistics: authenticated refreshes to RPC server</td>
</tr>
<tr>
<td>nfs_client_rpc_calls</td>
<td>NFS Client RPC statistics: calls</td>
</tr>
<tr>
<td>nfs_client_rpc_retrans</td>
<td>NFS Client RPC statistics: re-transmissions</td>
</tr>
<tr>
<td>nfs_server_file_anon</td>
<td>NFS Server file statistics: anonymous access</td>
</tr>
<tr>
<td>nfs_server_file_lookup</td>
<td>NFS Server file statistics: look-ups</td>
</tr>
<tr>
<td>nfs_server_file_ncachedir</td>
<td>NFS Server file statistics: ncachedir</td>
</tr>
<tr>
<td>nfs_server_file_stale</td>
<td>NFS Server file statistics: stale files</td>
</tr>
<tr>
<td>nfs_server_packet_packets</td>
<td>NFS Server packet statistics: packets</td>
</tr>
<tr>
<td>nfs_server_packet_tcp</td>
<td>NFS Server packet statistics: TCP/IP packets</td>
</tr>
<tr>
<td>nfs_server_packet_tcpconn</td>
<td>NFS Server packet statistics: TCP/IP connections</td>
</tr>
<tr>
<td>nfs_server_packet_udp</td>
<td>NFS Server packet statistics: UDP packets</td>
</tr>
<tr>
<td>nfs_server_reply_hits</td>
<td>NFS Server reply statistics: hits</td>
</tr>
<tr>
<td>nfs_server_reply_misses</td>
<td>NFS Server reply statistics: misses</td>
</tr>
<tr>
<td>nfs_server_reply_nocache</td>
<td>NFS Server reply statistics: no cache</td>
</tr>
<tr>
<td>nfs_server_rpc_badauth</td>
<td>NFS Server RPC statistics: bad authentication</td>
</tr>
<tr>
<td>nfs_server_rpc_badcalls</td>
<td>NFS Server RPC statistics: bad RPC requests</td>
</tr>
<tr>
<td>nfs_server_rpc_badclnt</td>
<td>NFS Server RPC statistics: badclnt</td>
</tr>
<tr>
<td>nfs_server_rpc_calls</td>
<td>NFS Server RPC statistics: all calls to NFS and NLM</td>
</tr>
</tbody>
</table>

...continues
Table G.1.2: NFS Metrics

<table>
<thead>
<tr>
<th>NFS Metric</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>nfs_server_rpc_xdrcall</td>
<td>NFS Server RPC statistics: malformed XDR calls</td>
</tr>
<tr>
<td>nfs_v3_client_access</td>
<td>NFSv3 client statistics: access</td>
</tr>
<tr>
<td>nfs_v3_client_fsinfo</td>
<td>NFSv3 client statistics: static file system information</td>
</tr>
<tr>
<td>nfs_v3_client_fsstat</td>
<td>NFSv3 client statistics: dynamic file system status</td>
</tr>
<tr>
<td>nfs_v3_client_getattr</td>
<td>NFSv3 client statistics: file system attributes</td>
</tr>
<tr>
<td>nfs_v3_client_lookup</td>
<td>NFSv3 client statistics: lookup</td>
</tr>
<tr>
<td>nfs_v3_client_pathconf</td>
<td>NFSv3 client statistics: configuration path</td>
</tr>
<tr>
<td>nfs_v3_client_read</td>
<td>NFSv3 client statistics: reads</td>
</tr>
<tr>
<td>nfs_v3_client_total</td>
<td>NFSv3 client statistics: total</td>
</tr>
<tr>
<td>nfs_v3_server_access</td>
<td>NFSv3 server statistics: access</td>
</tr>
<tr>
<td>nfs_v3_server_create</td>
<td>NFSv3 server statistics: create</td>
</tr>
<tr>
<td>nfs_v3_server_fsinfo</td>
<td>NFSv3 server statistics: static file system information</td>
</tr>
<tr>
<td>nfs_v3_server_fsstat</td>
<td>NFSv3 server statistics: dynamic file system information</td>
</tr>
<tr>
<td>nfs_v3_server_getattr</td>
<td>NFSv3 server statistics: file system attributes gets</td>
</tr>
<tr>
<td>nfs_v3_server_lookup</td>
<td>NFSv3 server statistics: file name look-ups</td>
</tr>
<tr>
<td>nfs_v3_server_mkdir</td>
<td>NFSv3 server statistics: directory creation</td>
</tr>
<tr>
<td>nfs_v3_server_null</td>
<td>NFSv3 server statistics: null operations</td>
</tr>
<tr>
<td>nfs_v3_server_pathconf</td>
<td>NFSv3 server statistics: retrieve POSIX information</td>
</tr>
<tr>
<td>nfs_v3_server_read</td>
<td>NFSv3 server statistics: reads</td>
</tr>
<tr>
<td>nfs_v3_server_readdirplus</td>
<td>NFSv3 server statistics: READDIRPLUS procedures</td>
</tr>
<tr>
<td>nfs_v3_server_readlink</td>
<td>NFSv3 server statistics: Symbolic link reads</td>
</tr>
<tr>
<td>nfs_v3_server_setattr</td>
<td>NFSv3 server statistics: file system attribute sets</td>
</tr>
<tr>
<td>nfs_v3_server_total</td>
<td>NFSv3 server statistics: total</td>
</tr>
<tr>
<td>nfs_v3_server_write</td>
<td>NFSv3 server statistics: writes</td>
</tr>
</tbody>
</table>

G.1.3 InfiniBand Metrics
The available InfiniBand metrics are displayed in table G.1.3.

Table G.1.3: InfiniBand Metrics

<table>
<thead>
<tr>
<th>InfiniBand Metric</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>SymbolErrorCount</td>
<td>Total number of minor link errors detected on one or more physical lanes.</td>
</tr>
<tr>
<td>LinkErrorRecoveryCount</td>
<td>Total number of times the Port Training state machine has successfully completed the link error recovery process.</td>
</tr>
</tbody>
</table>

...continues
<table>
<thead>
<tr>
<th>IB Metric</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>LinkDownedCounter</td>
<td>Total number of times the Port Training state machine has failed the link error recovery process and downed the link.</td>
</tr>
<tr>
<td>PortRcvErrors</td>
<td>Total number of packets containing an error that were received on the port.</td>
</tr>
<tr>
<td>PortRcvRemotePhysicalErrors</td>
<td>Total number of packets marked with the EBP delimiter received on the port.</td>
</tr>
<tr>
<td>PortRcvSwitchRelayErrors</td>
<td>Total number of packets received on the port that were discarded because they could not be forwarded by the switch relay.</td>
</tr>
<tr>
<td>PortXmitDiscards</td>
<td>Total number of outbound packets discarded by the port because the port is down or congested.</td>
</tr>
<tr>
<td>PortXmitConstrainErrors</td>
<td>Total number of packets not transmitted from the switch physical port.</td>
</tr>
<tr>
<td>PortRcvConstraintErrors</td>
<td>Total number of packets received on the switch physical port that are discarded.</td>
</tr>
<tr>
<td>LocalLinkIntegrityErrors</td>
<td>The number of times that the count of local physical errors exceeded the threshold specified by LocalPhyErrors.</td>
</tr>
<tr>
<td>ExcessiveBufferOverrunError</td>
<td>The number of times that OverrunErrors consecutive flow control update periods occurred, each having at least one overrun error.</td>
</tr>
<tr>
<td>QP1Dropped</td>
<td>Drops on the lower priority QP1 interconnect</td>
</tr>
<tr>
<td>VL15Dropped</td>
<td>Drops in the highest priority virtual lane 15</td>
</tr>
<tr>
<td>PortXmitData</td>
<td>Total number of data octets, divided by 4 (lanes), transmitted on all VLs. This is a 64-bit counter.</td>
</tr>
<tr>
<td>PortRcvData</td>
<td>Total number of data octets, divided by 4 (lanes), received on all VLs. This is a 64-bit counter.</td>
</tr>
<tr>
<td>PortXmitPkts</td>
<td>Total number of packets transmitted on all VLs from this port, including packets with errors, and excluding link packets.</td>
</tr>
<tr>
<td>PortRcvPkts</td>
<td>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.</td>
</tr>
<tr>
<td>PortXmitWait</td>
<td>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).</td>
</tr>
</tbody>
</table>

**G.1.4 Monitoring System Metrics**

The monitoring system metrics correspond to metrics for the monitoring data storage itself. These are shown in table G.1.4.
Table G.1.4: Monitoring System Metrics

<table>
<thead>
<tr>
<th>Monitoring System Metric</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Mon::*CacheGather::*handled</td>
<td>Cache gathers handled/s</td>
</tr>
<tr>
<td>Mon::*CacheGather::*miss</td>
<td>Cache gathers missed/s</td>
</tr>
<tr>
<td>Mon::*DataProcessor::*handled</td>
<td>Data processors handled/s</td>
</tr>
<tr>
<td>Mon::*DataProcessor::*miss</td>
<td>Data processors missed/s</td>
</tr>
<tr>
<td>Mon::*DataTranslator::*handled</td>
<td>Data translators handled/s</td>
</tr>
<tr>
<td>Mon::*DataTranslator::*miss</td>
<td>Data translators missed/s</td>
</tr>
<tr>
<td>Mon::*EntityMeasurableCache::*handled</td>
<td>Measurable cache handled/s</td>
</tr>
<tr>
<td>Mon::*EntityMeasurableCache::*miss</td>
<td>Measurable cache missed/s</td>
</tr>
<tr>
<td>Mon::*MeasurableBroker::*handled</td>
<td>Measurable broker handled/s</td>
</tr>
<tr>
<td>Mon::*MeasurableBroker::*miss</td>
<td>Measurable broker missed/s</td>
</tr>
<tr>
<td>Mon::*OOB::*TaskService::*handled</td>
<td>Out-of-band task service handled/s</td>
</tr>
<tr>
<td>Mon::*OOB::*TaskService::*miss</td>
<td>Out-of-band task service missed/s</td>
</tr>
<tr>
<td>Mon::*Replicate::*Collector::*handled</td>
<td>Replication collection handled/s</td>
</tr>
<tr>
<td>Mon::*Replicate::*Collector::*miss</td>
<td>Replication collection missed/s</td>
</tr>
<tr>
<td>Mon::*Replicate::*Combiner::*handled</td>
<td>Replication combiner handled/s</td>
</tr>
<tr>
<td>Mon::*Replicate::*Combiner::*miss</td>
<td>Replication combiner missed/s</td>
</tr>
<tr>
<td>Mon::*RepositoryAllocator::*handled</td>
<td>Repository allocator handled/s</td>
</tr>
<tr>
<td>Mon::*RepositoryAllocator::*miss</td>
<td>Repository allocator missed/s</td>
</tr>
<tr>
<td>Mon::*RepositoryTrim::*handled</td>
<td>Repository trim handled/s</td>
</tr>
<tr>
<td>Mon::*RepositoryTrim::*miss</td>
<td>Repository trim missed/s</td>
</tr>
<tr>
<td>Mon::*Storage::*Engine::*elements</td>
<td>Storage engine data elements, in total</td>
</tr>
<tr>
<td>Mon::*Storage::*Engine::*size</td>
<td>Storage engine size, in bytes</td>
</tr>
<tr>
<td>Mon::*Storage::*Engine::*usage</td>
<td>Storage engine usage</td>
</tr>
<tr>
<td>Mon::*Storage::*Message::*elements</td>
<td>Storage message data elements, in total</td>
</tr>
<tr>
<td>Mon::*Storage::*Message::*size</td>
<td>Storage message size in bytes</td>
</tr>
<tr>
<td>Mon::*Storage::*Message::*usage</td>
<td>Storage message usage</td>
</tr>
<tr>
<td>Mon::*Storage::*RepositoryId::*elements</td>
<td>Storage repository ID data elements, in total</td>
</tr>
<tr>
<td>Mon::*Storage::*RepositoryId::*size</td>
<td>Storage repository ID, size, in bytes</td>
</tr>
<tr>
<td>Mon::*Storage::*RepositoryId::*usage</td>
<td>Repository ID usage</td>
</tr>
<tr>
<td>Mon::*TaskInitializer::*handled</td>
<td>Task initializer handled/s</td>
</tr>
<tr>
<td>Mon::*TaskInitializer::*miss</td>
<td>Task initializer missed/s</td>
</tr>
<tr>
<td>Mon::*TaskSampler::*handled</td>
<td>Task sampler handled/s</td>
</tr>
<tr>
<td>Mon::*TaskSampler::*miss</td>
<td>Task sampler missed/s</td>
</tr>
<tr>
<td>Mon::*Trigger::*Actuator::*handled</td>
<td>Trigger actuators handled/s</td>
</tr>
<tr>
<td>Mon::*Trigger::*Actuator::*miss</td>
<td>Trigger actuators missed/s</td>
</tr>
<tr>
<td>Mon::*Trigger::*Dispatcher::*handled</td>
<td>Trigger dispatchers handled/s</td>
</tr>
</tbody>
</table>

...continues
Table G.1.4: Monitoring System Metrics...continued

<table>
<thead>
<tr>
<th>Monitoring System Metric</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Mon::Trigger::Dispatcher::miss</td>
<td>Trigger dispatchers missed/s</td>
</tr>
<tr>
<td>Prometheus::DataTranslator::handled</td>
<td>DataTranslator queries handled. The DataTranslator is a translation layer between PromQL format and the cluster manager format sampling.</td>
</tr>
<tr>
<td>Prometheus::DataTranslator::miss</td>
<td>DataTranslator misses</td>
</tr>
</tbody>
</table>

G.1.5 GPU Metrics

The data producer (section 13.2.10) for the GPU metrics of this section is GPUSampler.

There were GPU metrics described earlier on in table G.1.1, These were cluster overview metrics about GPUs, and were provided by the ClusterTotal data producer.

However, the NVIDIA GPU metrics of this section, as the GPUSampler data producer name suggests, is about gathering the sampled GPU data from the devices themselves.

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 G.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 G.2.1.

Table G.1.5: GPU Metrics

<table>
<thead>
<tr>
<th>GPU Metric</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>gpu_ecc_dbe_agg</td>
<td>Total double bit aggregate ECC errors</td>
</tr>
<tr>
<td>gpu_ecc_dbe_vol</td>
<td>Total double bit volatile ECC errors</td>
</tr>
<tr>
<td>gpu_ecc_sbe_agg</td>
<td>Total single bit aggregate ECC errors</td>
</tr>
<tr>
<td>gpu_ecc_sbe_vol</td>
<td>Total single bit volatile ECC errors</td>
</tr>
<tr>
<td>gpu_enforced_power_limit</td>
<td>GPU enforced power limit</td>
</tr>
<tr>
<td>gpu_mem_clock</td>
<td>GPU memory clock (also for AMD GPU)</td>
</tr>
<tr>
<td>gpu_mem_total</td>
<td>GPU framebuffer size (also for AMD GPU)</td>
</tr>
<tr>
<td>gpu_mem_used</td>
<td>Amount of GPU memory used (also for AMD GPU)</td>
</tr>
</tbody>
</table>

...continues
### G.1.5 GPU Metrics and Their Parameters

<table>
<thead>
<tr>
<th>GPU Metric</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>gpu_mem_utilization</td>
<td>GPU memory used percentage</td>
</tr>
<tr>
<td>gpu_nvlink_total_bandwidth</td>
<td>Total NVLink bandwidth used</td>
</tr>
<tr>
<td>gpu_power_management_limit</td>
<td>GPU power management limit</td>
</tr>
<tr>
<td>gpu_power_usage</td>
<td>GPU power usage (also for AMD GPU)</td>
</tr>
<tr>
<td>gpu_profiling_fp16_active</td>
<td>ratio of cycles the FP16 pipe is active</td>
</tr>
<tr>
<td>gpu_profiling_fp64_active</td>
<td>ratio of cycles the FP64 pipe is active</td>
</tr>
<tr>
<td>gpu_profiling_pipe_tensor_active</td>
<td>ratio of cycles the tensor (HMMA) pipe is active (A100 only)</td>
</tr>
<tr>
<td>gpu_shutdown_temp</td>
<td>GPU shutdown temperature</td>
</tr>
<tr>
<td>gpu_slowdown_temp</td>
<td>GPU slowdown temperature</td>
</tr>
<tr>
<td>gpu_sm_clock</td>
<td>GPU shader multiprocessor clock (also for AMD GPU)</td>
</tr>
<tr>
<td>gpu_temperature</td>
<td>GPU temperature</td>
</tr>
<tr>
<td>gpu_temperature_edge</td>
<td>GPU edge temperature (AMD GPU only)</td>
</tr>
<tr>
<td>gpu_temperature_junction</td>
<td>GPU junction temperature (AMD GPU only)</td>
</tr>
<tr>
<td>gpu_temperature_memory</td>
<td>GPU memory temperature (AMD GPU only)</td>
</tr>
<tr>
<td>gpu_utilization</td>
<td>Average GPU utilization percentage. This takes one of the following parameters:</td>
</tr>
<tr>
<td></td>
<td>• <strong>gpu_utilization:gpu&lt;number&gt;</strong>: 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</td>
</tr>
<tr>
<td></td>
<td>• <strong>gpu_utilization:average</strong>: 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</td>
</tr>
<tr>
<td></td>
<td>• <strong>gpu_utilization:compute_instance&lt;number&gt;</strong>: the GPU utilization percentage for a MIG GPU instance 0 (A100 only). The compute instance &lt;number&gt; can be from 0 up to 6 for the A100 MIGs.</td>
</tr>
</tbody>
</table>

### G.1.6 Job Metrics

Job metrics are introduced in section 14.1.

**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 G.1.6.1: Basic Job Metrics

<table>
<thead>
<tr>
<th>Job Metric</th>
<th>Description</th>
<th>Cgroup Source File</th>
</tr>
</thead>
<tbody>
<tr>
<td>blkio.time:&lt;device&gt;</td>
<td>Time job had I/O access to device</td>
<td>blkio.time_recursive</td>
</tr>
<tr>
<td>blkio.sectors:&lt;device&gt;</td>
<td>Sectors transferred to or from specific devices by a cgroup</td>
<td>blkio.sectors_recursive</td>
</tr>
<tr>
<td>blkio.io_service_read:&lt;device&gt;</td>
<td>Bytes read</td>
<td>blkio.io_service_bytes_recursive</td>
</tr>
<tr>
<td>blkio.io_service_write:&lt;device&gt;</td>
<td>Bytes written</td>
<td>blkio.io_service_bytes_recursive</td>
</tr>
<tr>
<td>blkio.io_service_sync:&lt;device&gt;</td>
<td>Bytes transferred synchronously</td>
<td>blkio.io_service_bytes_recursive</td>
</tr>
<tr>
<td>blkio.io_service_async:&lt;device&gt;</td>
<td>Bytes transferred asynchronously</td>
<td>blkio.io_service_bytes_recursive</td>
</tr>
<tr>
<td>blkio.io_wait_time_read:&lt;device&gt;</td>
<td>Total time spent waiting for service in the scheduler queues for I/O read operations</td>
<td>blkio.io_wait_time_recursive</td>
</tr>
<tr>
<td>blkio.io_wait_time_write:&lt;device&gt;</td>
<td>Total time spent waiting for service in the scheduler queues for I/O write operations</td>
<td>blkio.io_wait_time_recursive</td>
</tr>
<tr>
<td>blkio.io_wait_time_sync:&lt;device&gt;</td>
<td>Total time spent waiting for service in the scheduler queues for I/O synchronous operations</td>
<td>blkio.io_wait_time_recursive</td>
</tr>
<tr>
<td>blkio.io_wait_time_async:&lt;device&gt;</td>
<td>Total time spent waiting for service in the scheduler queues for I/O asynchronous operations</td>
<td>blkio.io_wait_time_recursive</td>
</tr>
<tr>
<td>cpuacct.usage</td>
<td>Total CPU time consumed by all job processes</td>
<td>cpuacct.usage</td>
</tr>
<tr>
<td>cpuacct.stat.user</td>
<td>User CPU time consumed by all job processes</td>
<td>cpuacct.stat</td>
</tr>
<tr>
<td>cpuacct.stat.system</td>
<td>System CPU time consumed by all job processes</td>
<td>cpuacct.stat</td>
</tr>
<tr>
<td>memory.usage</td>
<td>Total current memory usage</td>
<td>memory.usage_in_bytes</td>
</tr>
<tr>
<td>memory.memsv.usage</td>
<td>Sum of current memory plus swap space usage</td>
<td>memory.memsv.usage_in_bytes</td>
</tr>
<tr>
<td>memory.memsv.max_usage</td>
<td>Maximum amount of memory and swap space used</td>
<td>memory.memsv.max_usage_in_bytes</td>
</tr>
<tr>
<td>memory.failcnt</td>
<td>How often the memory limit has reached the value set in memory.limit_in_bytes</td>
<td>memory.failcnt</td>
</tr>
</tbody>
</table>

...continues
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:

```
[bright92->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
```

```
[bright92->monitoring->setup[JobSampler]->jobmetricsettings]% set enableadvancedmetrics yes
[bright92->monitoring->setup*[JobSampler*]->jobmetricsettings*]% commit
```

The advanced job metrics are:

```
Table G.1.6.2: Advanced Job Metrics

<table>
<thead>
<tr>
<th>Advanced Job Metric</th>
<th>Description</th>
<th>Cgroup Source File</th>
</tr>
</thead>
<tbody>
<tr>
<td>blkio.io_service_time_read:&lt;device&gt;</td>
<td>Total time between request dispatch and request completion according to CFQ scheduler for I/O read operations</td>
<td>blkio.io_service_time_recursive</td>
</tr>
</tbody>
</table>
```

...continues
### Advanced Job Metric Description

<table>
<thead>
<tr>
<th>Metric Code</th>
<th>Description</th>
<th>Cgroup Source File</th>
</tr>
</thead>
<tbody>
<tr>
<td>blkio.io_service_time_write:&lt;device&gt;</td>
<td>Total time between request dispatch and request completion according to CFQ scheduler for I/O write operations</td>
<td>blkio.io_service_time_recursive</td>
</tr>
<tr>
<td>blkio.io_service_time_sync:&lt;device&gt;</td>
<td>Total time between request dispatch and request completion according to CFQ scheduler for I/O synchronous operations</td>
<td>blkio.io_service_time_recursive</td>
</tr>
<tr>
<td>blkio.io_service_time_async:&lt;device&gt;</td>
<td>Total time between request dispatch and request completion according to CFQ scheduler for I/O asynchronous operations</td>
<td>blkio.io_service_time_recursive</td>
</tr>
<tr>
<td>blkio.io_serviced_read:&lt;device&gt;</td>
<td>Read I/O operations</td>
<td>blkio.io_serviced_recursive</td>
</tr>
<tr>
<td>blkio.io_serviced_write:&lt;device&gt;</td>
<td>Write I/O operations</td>
<td>blkio.io_serviced_recursive</td>
</tr>
<tr>
<td>blkio.io_serviced_sync:&lt;device&gt;</td>
<td>Synchronous I/O operations</td>
<td>blkio.io_serviced_recursive</td>
</tr>
<tr>
<td>blkio.io_serviced_async:&lt;device&gt;</td>
<td>Asynchronous I/O operations</td>
<td>blkio.io_serviced_recursive</td>
</tr>
<tr>
<td>blkio.io_merged_read:&lt;device&gt;</td>
<td>Number of BIOS requests merged into requests for I/O read operations</td>
<td>blkio.io_merged_recursive</td>
</tr>
<tr>
<td>blkio.io_merged_write:&lt;device&gt;</td>
<td>Number of BIOS requests merged into requests for I/O write operations</td>
<td>blkio.io_merged_recursive</td>
</tr>
<tr>
<td>blkio.io_merged_sync:&lt;device&gt;</td>
<td>Number of BIOS requests merged into requests for I/O synchronous operations</td>
<td>blkio.io_merged_recursive</td>
</tr>
<tr>
<td>blkio.io_merged_async:&lt;device&gt;</td>
<td>Number of BIOS requests merged into requests for I/O asynchronous operations</td>
<td>blkio.io_merged_recursive</td>
</tr>
<tr>
<td>blkio.io_queued_read:&lt;device&gt;</td>
<td>Number of requests queued for I/O read operations</td>
<td>blkio.io_queued_recursive</td>
</tr>
<tr>
<td>blkio.io_queued_write:&lt;device&gt;</td>
<td>Number of requests queued for I/O write operations</td>
<td>blkio.io_queued_recursive</td>
</tr>
<tr>
<td>blkio.io_queued_sync:&lt;device&gt;</td>
<td>Number of requests queued for I/O synchronous operations</td>
<td>blkio.io_queued_recursive</td>
</tr>
</tbody>
</table>

...continues
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 memory hierarchy that contains the memory cgroup of job
  - `memory`

- `memory.hierarchical\_memsw_limit`
  - Memory plus swap limit for memory hierarchy that contains the memory cgroup of job
  - `memory`

---

G.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 13.2.10) for the IPMI metrics is the `ipmi` data producer.

<table>
<thead>
<tr>
<th>IPMI Metric</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Current_&lt;number&gt;**</td>
<td>Current seen by BMC sensor &lt;number&gt;, in amps (file: sample_ipmi)</td>
</tr>
</tbody>
</table>

...continues
### Table G.1.7: IPMI Metrics...continued

<table>
<thead>
<tr>
<th>IPMI Metric</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Exhaust_Temp</td>
<td>Exhaust temperature, in Celsius</td>
</tr>
<tr>
<td>FETDRV_PG</td>
<td>PowerEdge voltage sensor on IPMI board</td>
</tr>
<tr>
<td>Fan&lt;number&gt;._RPM</td>
<td>RPM of Fan&lt;number&gt; as seen by BMC (file: sample_ipmi)</td>
</tr>
<tr>
<td>Fan_Redundancy</td>
<td>fan redundancy status</td>
</tr>
<tr>
<td>Inlet_Temp**</td>
<td>Inlet Temperature, in Celsius (file: sample_ipmi)</td>
</tr>
<tr>
<td>M01_VDDQ_PG</td>
<td>PowerEdge CPU voltage sensor M01_VDDQ_PG</td>
</tr>
<tr>
<td>M01_VTT_PG</td>
<td>PowerEdge CPU voltage sensor M01_VTT_PG</td>
</tr>
<tr>
<td>M23_VDDQ_PG</td>
<td>PowerEdge CPU voltage sensor M23_VDDQ_PG</td>
</tr>
<tr>
<td>M23_VTT_PG</td>
<td>PowerEdge CPU voltage sensor M23_VTT_PG</td>
</tr>
<tr>
<td>NDC_PG</td>
<td>PowerEdge system board voltage sensor</td>
</tr>
<tr>
<td>PFault_Fail_Safe</td>
<td>PowerEdge sensor</td>
</tr>
<tr>
<td>PLL_PG</td>
<td>PowerEdge sensor</td>
</tr>
<tr>
<td>PS1_PG_Fail</td>
<td>PowerEdge sensor</td>
</tr>
<tr>
<td>PS2_PG_Fail</td>
<td>PowerEdge sensor</td>
</tr>
<tr>
<td>Pwr_Consumption</td>
<td>Power consumed by BMC, in watts (file: sample_ipmi)</td>
</tr>
<tr>
<td>Temp</td>
<td>Temperature, in Celsius</td>
</tr>
<tr>
<td>VSA_PG</td>
<td>PowerEdge voltage sensor</td>
</tr>
<tr>
<td>VTT_PG</td>
<td>PowerEdge voltage sensor</td>
</tr>
<tr>
<td>Voltage_&lt;number&gt;*</td>
<td>Voltage seen by BMC sensor number, in Volts (file: sample_ipmi)</td>
</tr>
</tbody>
</table>

** Standalone scripts, not built-ins.

If sampling from a head node, the script is in directory: /cm/local/apps/cmd/scripts/metrics/

For regular nodes, the script is in directory: /cm/images/default-image/cm/local/apps/cmd/scripts/metrics/

### G.1.8 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 13.2.10) for SMART metrics is the smart data producer.
Table G.1.7: SMART Metrics

<table>
<thead>
<tr>
<th>SMART Metric</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Command_Timeout</td>
<td>Command timeout</td>
</tr>
<tr>
<td>Current_Pending_Sector</td>
<td>Current pending sectors</td>
</tr>
<tr>
<td>Hardware_ECC_Recovered</td>
<td>Hardware ECC recovered</td>
</tr>
<tr>
<td>Offline_Uncorrectable</td>
<td>Uncorrectable sectors</td>
</tr>
<tr>
<td>Raw_Read_Error_Rate</td>
<td>Raw read error rate</td>
</tr>
<tr>
<td>Reallocated_Sector.Ct</td>
<td>Reallocated sectors count</td>
</tr>
<tr>
<td>Reported_Uncorrect</td>
<td>Reported uncorrectable errors</td>
</tr>
<tr>
<td>UDMA_CRC_Error_Count</td>
<td>UDMA CRC errors</td>
</tr>
</tbody>
</table>

G.1.9 Prometheus Metrics

Prometheus metrics are introduced in section 15.2.

The data producers for Prometheus metrics are JobSampler, JobMetadataSampler, and some others.

Table G.1.9: Prometheus Metrics

<table>
<thead>
<tr>
<th>Prometheus Metric</th>
<th>Description (for a job, unless asterisked)</th>
</tr>
</thead>
<tbody>
<tr>
<td>job_blkio_io_merges</td>
<td>Number of BIOS requests merged into requests for I/O operations by a cgroup</td>
</tr>
<tr>
<td>job_blkio_io_queued</td>
<td>Number of requests queued for I/O operations by a cgroup</td>
</tr>
<tr>
<td>job_blkio_io_service_bytes</td>
<td>Reports the number of bytes transferred to or from specific devices by a cgroup as seen by the CFQ scheduler</td>
</tr>
<tr>
<td>job_blkio_io_service_bytes_total</td>
<td>Reports the number of bytes transferred to or from specific devices by a cgroup as seen by the CFQ scheduler (for all jobs)</td>
</tr>
<tr>
<td>job_blkio_io_service_time_seconds</td>
<td>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</td>
</tr>
<tr>
<td>job_blkio_io_serviced</td>
<td>Reports the number of I/O operations performed on specific devices by a cgroup as seen by the CFQ scheduler</td>
</tr>
<tr>
<td>job_blkio_io_wait_time_seconds</td>
<td>Reports the total time I/O operations on specific devices by a cgroup spent waiting for service in the scheduler queues</td>
</tr>
</tbody>
</table>

...continues
Table G.1.9: Prometheus Metrics...continued

<table>
<thead>
<tr>
<th>Prometheus Metric</th>
<th>Description (for a job, unless asterisked)</th>
</tr>
</thead>
<tbody>
<tr>
<td>job_blkio_sectors</td>
<td>Reports the number of sectors transferred to or from specific devices by a cgroup</td>
</tr>
<tr>
<td>job_blkio_time_seconds</td>
<td>Reports the time that a cgroup had I/O access to specific devices</td>
</tr>
<tr>
<td>job_cpuacct_stat_system</td>
<td>System CPU time consumed by processes</td>
</tr>
<tr>
<td>job_cpuacct_stat_user</td>
<td>User CPU time consumed by processes</td>
</tr>
<tr>
<td>job_cpuacct_usage_seconds</td>
<td>CPU usage time consumed</td>
</tr>
<tr>
<td>job_memory_active_anon_bytes</td>
<td>Anonymous and swap cache on active least-recently-used (LRU) list, including tmpfs (shmem), in bytes</td>
</tr>
<tr>
<td>job_memory_active_file_bytes</td>
<td>File-backed memory on active LRU list, in bytes</td>
</tr>
<tr>
<td>job_memory_cache_bytes</td>
<td>Page cache, including tmpfs (shmem), in bytes</td>
</tr>
<tr>
<td>job_memory_failcnt</td>
<td>Reports the number of times that the memory limit has reached the value set in memory.limit_in_bytes</td>
</tr>
<tr>
<td>job_memory_hierarchical_memory_limit_bytes</td>
<td>Memory limit for the hierarchy that contains the memory cgroup, in bytes</td>
</tr>
<tr>
<td>job_memory_hierarchical_memsw_limit_bytes</td>
<td>Memory plus swap limit for the hierarchy that contains the memory cgroup, in bytes</td>
</tr>
<tr>
<td>job_memory_inactive_anon_bytes</td>
<td>Anonymous and swap cache on inactive LRU list, including tmpfs (shmem), in bytes</td>
</tr>
<tr>
<td>job_memory_inactive_file_bytes</td>
<td>File-backed memory on inactive LRU list, in bytes</td>
</tr>
<tr>
<td>job_memory_mapped_file_bytes</td>
<td>Size of memory-mapped mapped files, including tmpfs (shmem), in bytes</td>
</tr>
<tr>
<td>job_memory_memsw_failcnt</td>
<td>Reports the number of times that the memory plus swap space limit has reached the value set in memory.memsw.limit_in_bytes</td>
</tr>
<tr>
<td>job_memory_memsw_max_usage_bytes</td>
<td>Reports the maximum amount of memory and swap space used by processes in the cgroup, in bytes</td>
</tr>
<tr>
<td>job_memory_memsw_usage_bytes</td>
<td>Reports the sum of current memory usage plus swap space used by processes in the cgroup, in bytes</td>
</tr>
<tr>
<td>job_memory_pgpgin_bytes</td>
<td>Number of pages paged into memory</td>
</tr>
<tr>
<td>job_memory_pgpgout_bytes</td>
<td>Number of pages paged out of memory</td>
</tr>
</tbody>
</table>

...continues
Table G.1.9: Prometheus Metrics—continued

<table>
<thead>
<tr>
<th>Prometheus Metric</th>
<th>Description (for a job, unless asterisked)</th>
</tr>
</thead>
<tbody>
<tr>
<td>job_memory_rss_bytes</td>
<td>Anonymous and swap cache, not including tmpfs (shmem), in bytes</td>
</tr>
<tr>
<td>job_memory_swap_bytes</td>
<td>Swap usage, in bytes</td>
</tr>
<tr>
<td>job_memory_unevictable_bytes</td>
<td>Memory that cannot be reclaimed, in bytes,</td>
</tr>
<tr>
<td>job_memory_usage_bytes</td>
<td>Reports the total current memory usage by processes in the cgroup, in bytes</td>
</tr>
<tr>
<td>job_memory_usage_bytes_total</td>
<td>Reports the total current memory usage by processes in the cgroup, in bytes (for all jobs)*</td>
</tr>
<tr>
<td>job_metadata_allocated_cpu_cores</td>
<td>CPU cores used by a job by the user</td>
</tr>
<tr>
<td>job_metadata_allocated_gpus</td>
<td>GPUs used by a job by the user</td>
</tr>
<tr>
<td>job_metadata_is_running</td>
<td>Returns 1 if the job metadata sampler is running a job</td>
</tr>
<tr>
<td>job_metadata_is_waiting</td>
<td>Returns 1 if the job metadata sampler is waiting</td>
</tr>
<tr>
<td>job_metadata_num_cpus</td>
<td>Number of CPUs that the job runs on</td>
</tr>
<tr>
<td>job_metadata_num_nodes</td>
<td>Number of nodes that the job runs on</td>
</tr>
<tr>
<td>job_metadata_pending_jobs</td>
<td>Number of pending jobs for the user</td>
</tr>
<tr>
<td>job_metadata_running_jobs</td>
<td>Number of running jobs for the user</td>
</tr>
<tr>
<td>job_metadata_running_seconds</td>
<td>Time the job has run</td>
</tr>
<tr>
<td>job_metadata_waiting_seconds</td>
<td>Time the job has been waiting to run</td>
</tr>
<tr>
<td>users_job_effective_cpu_seconds:1w</td>
<td>CPU seconds used by users over the past week*</td>
</tr>
<tr>
<td>users_job_running_count:1w</td>
<td>Number of jobs run by users over the past week*</td>
</tr>
<tr>
<td>users_job_waiting_seconds:1w</td>
<td>Time users have been waiting for jobs to run over the past week*</td>
</tr>
<tr>
<td>users_job_wall_clock_seconds:1w</td>
<td>Time the jobs runs for users according to wall time over the past week*</td>
</tr>
<tr>
<td>users_job_wasted_cpu_seconds:1w</td>
<td>CPU time wasted during users jobs over the past week*</td>
</tr>
</tbody>
</table>

* total in the past 7x24x60x60 seconds, as measured at the time of sampling
G.1.10 Parameters For Metrics

Metrics have the parameters indicated by the left column in the following example:

**Example**

```
[bright02->monitoring->measurable[CPUUser]]% show
```

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>Class</td>
<td>CPU</td>
</tr>
<tr>
<td>Consolidator</td>
<td>default (ProcStat)</td>
</tr>
<tr>
<td>Cumulative</td>
<td>yes</td>
</tr>
<tr>
<td>Description</td>
<td>CPU time spent in user mode</td>
</tr>
<tr>
<td>Disabled</td>
<td>no (ProcStat)</td>
</tr>
<tr>
<td>Gap</td>
<td>0 (ProcStat)</td>
</tr>
<tr>
<td>Maximal age</td>
<td>0s (ProcStat)</td>
</tr>
<tr>
<td>Maximal samples</td>
<td>4,096 (ProcStat)</td>
</tr>
<tr>
<td>Maximum</td>
<td>0</td>
</tr>
<tr>
<td>Minimum</td>
<td>0</td>
</tr>
<tr>
<td>Name</td>
<td>CPUUser</td>
</tr>
<tr>
<td>Parameter</td>
<td></td>
</tr>
<tr>
<td>Producer</td>
<td>ProcStat</td>
</tr>
<tr>
<td>Revision</td>
<td></td>
</tr>
<tr>
<td>Type</td>
<td>Metric</td>
</tr>
<tr>
<td>Unit</td>
<td>jiffies</td>
</tr>
</tbody>
</table>

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 13.4.3 and 13.5.2
**G.1 Metrics And Their Parameters**

**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 default minimum value the y-axis maximum will take in graphs plotted in Bright View.

**Minimum**: the default maximum value the y-axis minimum will take in graphs plotted in Bright View.

**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`, `health check`, or `enum metric`.

**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 `%`

---

1To clarify the concept, if `maximum=3, minimum=0`, then a data-point with a y-value of 2 is plotted on a graph with the y-axis spanning from 0 to 3. However, if the data-point has a y-value of 4 instead, then it means the default y-axis maximum of 3 is re-sized to 4, and the y-axis will now span from 0 to 4.
G.2 Health Checks And Their Parameters

A list of health checks can be viewed, for example, for the head node, using cmsh as follows (section 13.5.3):

```
[bright92 ~]# cmsh -c "monitoring measurable; list healthcheck"
```

Regular health checks (section 13.2.4) are listed and described in section G.2.1. Redfish health checks are listed and described in section G.2.2.

G.2.1 Regular Health Checks

Table G.2.1: List Of Health Checks

<table>
<thead>
<tr>
<th>Name</th>
<th>Query (script response is PASS/FAIL)</th>
</tr>
</thead>
<tbody>
<tr>
<td>ManagedServicesOk*</td>
<td>Are CMDaemon-monitored services all OK? If the response is FAIL, then at least one of the services being monitored is failing. The latesthealthdata -v command (section 13.6.3) 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 123).</td>
</tr>
<tr>
<td>Mon::Storage</td>
<td>Is space available for the monitoring system metrics (section G.1.4)?</td>
</tr>
<tr>
<td>beegfs</td>
<td>Is BeeGFS properly connected as expected to all the nodes? Unreachable nodes and failed routes are listed in InfoMessages.</td>
</tr>
<tr>
<td>cmsh</td>
<td>Is cmsh available?</td>
</tr>
<tr>
<td>cuda-dcgm</td>
<td>Is cuda-dcgm available?</td>
</tr>
<tr>
<td>defaultgateway</td>
<td>Is there a default gateway available?</td>
</tr>
<tr>
<td>dellnss</td>
<td>If running, is the Dell NFS Storage Solution healthy?</td>
</tr>
<tr>
<td>diskspace</td>
<td>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 927) instead.</td>
</tr>
</tbody>
</table>

...continued
Table G.2.1: List Of Health Checks...continued

<table>
<thead>
<tr>
<th>Name</th>
<th>Query (response is PASS/FAIL)</th>
</tr>
</thead>
<tbody>
<tr>
<td>disk space</td>
<td>less than 10% space = FAIL, severity 10</td>
</tr>
<tr>
<td>disk space</td>
<td>less than 30% space = FAIL, with severity levels as indicated:</td>
</tr>
<tr>
<td>space left</td>
<td>severity</td>
</tr>
<tr>
<td>10%</td>
<td>30</td>
</tr>
<tr>
<td>20%</td>
<td>20</td>
</tr>
<tr>
<td>30%</td>
<td>10</td>
</tr>
<tr>
<td>disk space</td>
<td>less than 20GB space = FAIL, severity 10</td>
</tr>
<tr>
<td>disk space</td>
<td>less than 10GB space = FAIL, severity 20</td>
</tr>
<tr>
<td>disk space</td>
<td>less than 20% /var space = FAIL, severity 10</td>
</tr>
<tr>
<td>disk space</td>
<td>less than 10% /var space = FAIL, severity 20</td>
</tr>
</tbody>
</table>

...continued
<table>
<thead>
<tr>
<th>Name</th>
<th>Query (response is PASS/FAIL)</th>
</tr>
</thead>
<tbody>
<tr>
<td>dmesg</td>
<td>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</td>
</tr>
<tr>
<td>docker</td>
<td>Is Docker running OK? Checks for Docker server availability and corruption, dead containers, proper endpoints</td>
</tr>
<tr>
<td>dockerregistry</td>
<td>Is the Docker registry running OK? Checks registry endpoint and registry availability</td>
</tr>
<tr>
<td>exports</td>
<td>Are all filesystems as defined by the cluster management system exported?</td>
</tr>
<tr>
<td>etcd</td>
<td>Are the core etcd processes of Kubernetes running OK? Checks endpoints and interfaces</td>
</tr>
<tr>
<td>failedprejob</td>
<td>Are there failed prejob health checks(section 7.8.2)? Here: yes = FAIL. By default, the job ID is saved under /cm/local/apps/&lt;scheduler&gt;/var:/</td>
</tr>
<tr>
<td></td>
<td>• On FAIL, in failedprejobs.</td>
</tr>
<tr>
<td></td>
<td>• On PASS, in allprejobs</td>
</tr>
<tr>
<td></td>
<td>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.</td>
</tr>
<tr>
<td>failover</td>
<td>Is the failover status OK?</td>
</tr>
<tr>
<td>gpu_health_driver∥</td>
<td>Is the driver-related subsystem OK?</td>
</tr>
<tr>
<td>gpu_health_hostengine</td>
<td>Is the host engine status, for all GPU devices on that node, OK?</td>
</tr>
<tr>
<td>gpu_health_inforom∥</td>
<td>Is the Inforom OK?</td>
</tr>
<tr>
<td>gpu_health_mcu∥</td>
<td>Is the microcontroller unit OK?</td>
</tr>
<tr>
<td>gpu_health_mem∥</td>
<td>Is the memory subsystem OK?</td>
</tr>
<tr>
<td>gpu_health_nvlink∥</td>
<td>Is the NVLINK system OK?</td>
</tr>
<tr>
<td>gpu_health_nvswitch_fatal∥</td>
<td>Is the NVswitch showing no fatal errors?</td>
</tr>
<tr>
<td>gpu_health_nvswitch_non_fatal∥</td>
<td>Is the NVswitch showing no non-fatal errors?</td>
</tr>
</tbody>
</table>

...continued
### Table G.2.1: List Of Health Checks...continued

<table>
<thead>
<tr>
<th>Name</th>
<th>Query (response is PASS/FAIL)</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>gpu_health_overall</code></td>
<td>Is the overall health OK?</td>
</tr>
<tr>
<td><code>gpu_health_pcie</code></td>
<td>Is the PCIe system OK?</td>
</tr>
<tr>
<td><code>gpu_health_pmu</code></td>
<td>Is the power management unit OK?</td>
</tr>
<tr>
<td><code>gpu_health_power</code></td>
<td>Is the power OK?</td>
</tr>
<tr>
<td><code>gpu_health_sm</code></td>
<td>Is the streaming multiprocessor OK?</td>
</tr>
<tr>
<td><code>gpu_health_thermal</code></td>
<td>Is the temperature OK?</td>
</tr>
<tr>
<td><code>hpraid</code></td>
<td>Are the HP Smart Array controllers OK?</td>
</tr>
<tr>
<td><code>ib</code></td>
<td>Is the InfiniBand Host Channel Adapter working properly?</td>
</tr>
<tr>
<td></td>
<td>A configuration file for this health check is at /cm/local/apps/cmd/scripts/healthchecks/configfiles/ib.py</td>
</tr>
<tr>
<td><code>interfaces</code></td>
<td>Are the interfaces all up and OK?</td>
</tr>
<tr>
<td><code>ipmihealth</code></td>
<td>Is the BMC (IPMI or iLO) health OK? Uses the script <code>sample_ipmi</code>.</td>
</tr>
<tr>
<td><code>kuberneteschildnode</code></td>
<td>Are all Kubernetes child nodes up?</td>
</tr>
<tr>
<td><code>kubernetescomponentstatus</code></td>
<td>Are all expected agents and services up and running for active nodes?</td>
</tr>
<tr>
<td><code>kubernetesnodesstatus</code></td>
<td>Is the status for all Kubernetes nodes OK?</td>
</tr>
<tr>
<td><code>kubernetespodsstatus</code></td>
<td>Is the status for all pods OK?</td>
</tr>
<tr>
<td><code>ldap</code></td>
<td>Can the ID of the user be looked up with LDAP?</td>
</tr>
<tr>
<td><code>lustre</code></td>
<td>Is the Lustre filesystem running OK?</td>
</tr>
<tr>
<td><code>megaraid</code></td>
<td>Are the MegaRAID controllers OK?</td>
</tr>
<tr>
<td></td>
<td>The proprietary MegaCli software from LSI (<a href="http://www.lsi.com">http://www.lsi.com</a>) is needed for this health check.</td>
</tr>
<tr>
<td><code>mounts</code></td>
<td>Are all mounts defined in the fstab OK?</td>
</tr>
<tr>
<td><code>mysql</code></td>
<td>Is the status and configuration of MySQL correct?</td>
</tr>
<tr>
<td><code>node-hardware-profile</code></td>
<td>Is the specified node’s hardware configuration during health check use unchanged?</td>
</tr>
<tr>
<td></td>
<td>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”</td>
</tr>
</tbody>
</table>
### Table G.2.1: List Of Health Checks...continued

<table>
<thead>
<tr>
<th>Name</th>
<th>Query (response is PASS/FAIL)</th>
</tr>
</thead>
<tbody>
<tr>
<td>ntp</td>
<td>Is NTP synchronization happening?</td>
</tr>
</tbody>
</table>
| 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-oomkiller.action

_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 957) can then go ahead and kill. Illegitimate processes are processes that should not be running on the node. An illegitimate process is at least one of the following, by default:

- not part of the workload manager service or its jobs
- not a root- or system-owned process
- in the state Z, T, W, or X. States are described in the `ps` man pages in the section on “PROCESS STATE CODES”

...continued
### Table G.2.1: List Of Health Checks...continued

<table>
<thead>
<tr>
<th>Name</th>
<th>Query (response is PASS/FAIL)</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>Rogue process criteria can be configured in the file <code>/cm/local/apps/cmd/scripts/healthchecks/configfiles/rogueprocess.py</code> 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: <code>cmsh -c &quot;device; imageupdate -c default -w&quot;</code> for the <code>default</code> category of nodes.</td>
</tr>
<tr>
<td>scheduler</td>
<td>Are the queue instances of all schedulers on a node healthy?</td>
</tr>
<tr>
<td>smart</td>
<td>Is the SMART response healthy? The severities can be configured in the file <code>/cm/local/apps/cmd/scripts/healthchecks/configfiles/smart.conf</code>. By default, if a drive does not support the SMART commands and results in a &quot;Smart command failed&quot; 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 a whitelist of the disks to be checked within <code>/cm/local/apps/cmd/scripts/healthchecks/smart</code>.</td>
</tr>
<tr>
<td>ssh2node</td>
<td>Is passwordless ssh root login, from head to a node that is up, working? Some details of its behavior are:</td>
</tr>
<tr>
<td></td>
<td>• If root ssh login to the head node has been disabled, then the health check fails on the head node.</td>
</tr>
<tr>
<td></td>
<td>• The healthcheck <code>ssh2node</code> only fails if ssh key-based access, to a node that is in the <code>UP</code> state, fails. The <code>UP</code> state is determined by whether CMDaemon is running on that node.</td>
</tr>
<tr>
<td></td>
<td>• If the regular node is in a <code>DOWN</code> 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 <code>PASS</code>. The idea here is to check key access, and decouple it from the node state.</td>
</tr>
</tbody>
</table>

...continued
Table G.2.1: List Of Health Checks...continued

<table>
<thead>
<tr>
<th>Name</th>
<th>Query (response is PASS/FAIL)</th>
</tr>
</thead>
<tbody>
<tr>
<td>swraid</td>
<td>Are the software RAID arrays healthy?</td>
</tr>
<tr>
<td>testhealthcheck</td>
<td>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.</td>
</tr>
</tbody>
</table>

* 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

**G.2.2 Redfish Healthchecks**

The available Redfish health checks are displayed in table G.2.2.

Table G.2.2: Redfish Healthchecks

<table>
<thead>
<tr>
<th>Redfish Healthcheck</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>chassis_health</td>
<td>Health status of chassis</td>
</tr>
<tr>
<td>cpu_health</td>
<td>Health status of processor</td>
</tr>
<tr>
<td>memory_health</td>
<td>Health status of memory</td>
</tr>
<tr>
<td>storage_health</td>
<td>Health status of storage</td>
</tr>
<tr>
<td>device_health</td>
<td>Health status of storage device</td>
</tr>
<tr>
<td>drive_health</td>
<td>Health status of storage drive</td>
</tr>
<tr>
<td>volume_health</td>
<td>Health status of storage volume</td>
</tr>
<tr>
<td>psu_health</td>
<td>Health status of power supply</td>
</tr>
<tr>
<td>fan_health</td>
<td>Health status of a fan</td>
</tr>
<tr>
<td>sensor_health</td>
<td>Health status of a sensor</td>
</tr>
<tr>
<td>pcie_health</td>
<td>Health status of PCIe device</td>
</tr>
<tr>
<td>manager_health</td>
<td>Health status of manager (e.g.: HPE iLO)</td>
</tr>
</tbody>
</table>
G.2 Health Checks And Their Parameters

G.2.3 Parameters For Health Checks

Health checks have the parameters indicated by the left column in the example below:

Example

```bash
[myheadnode->monitoring->measurable] % show cmsh
Parameter                      Value
-------------------------------- ----------------------------------------------
Class                           Internal
Consolidator                    - (cmsh)
Description                     Checks whether cmsh is available, i.e. can we use cmsh for the default cluster?
Disabled                        no (cmsh)
Gap                             0 (cmsh)
Maximal age                     0s (cmsh)
Maximal samples                 4,096 (cmsh)
Name                            cmsh
Parameter                       
Producer                        cmsh
Revision                        
Type                            HealthCheck
```

If the value is inherited from the producer, then it is shown in parentheses next to the value. An inherited value can be overwritten by setting it directly for the parameter of a measurable.

The parameters are a subset of the parameters for metrics described in section G.1.10.
G.3 Enummetrics

Table G.3: List Of Enummetrics

<table>
<thead>
<tr>
<th>Name</th>
<th>Query</th>
</tr>
</thead>
<tbody>
<tr>
<td>DeviceStatus</td>
<td>What is the status of the device? Possible values are:</td>
</tr>
<tr>
<td></td>
<td>• up</td>
</tr>
<tr>
<td></td>
<td>• down</td>
</tr>
<tr>
<td></td>
<td>• closed</td>
</tr>
<tr>
<td></td>
<td>• installing</td>
</tr>
<tr>
<td></td>
<td>• installer_failed</td>
</tr>
<tr>
<td></td>
<td>• installer_rebooting,</td>
</tr>
<tr>
<td></td>
<td>• installer_callinginit</td>
</tr>
<tr>
<td></td>
<td>• installer_unreachable</td>
</tr>
<tr>
<td></td>
<td>• installer_burning</td>
</tr>
<tr>
<td></td>
<td>• burning</td>
</tr>
<tr>
<td></td>
<td>• unknown</td>
</tr>
<tr>
<td></td>
<td>• opening</td>
</tr>
<tr>
<td></td>
<td>• going_down</td>
</tr>
<tr>
<td></td>
<td>• pending</td>
</tr>
<tr>
<td></td>
<td>• no data</td>
</tr>
</tbody>
</table>
G.4 Actions And Their Parameters

G.4.1 Actions

Table G.4.1: List Of Actions

<table>
<thead>
<tr>
<th>Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Drain</td>
<td>Allows no new processes on a compute node from the workload manager. This means that already running jobs are permitted to complete. Usage</td>
</tr>
<tr>
<td></td>
<td>Tip: Plan for undrain from another node becoming active</td>
</tr>
<tr>
<td>Send e-mail to</td>
<td>Sends mail using the mailserver that was set up during server configuration. Default destination is root@localhost. The e-mail address that it is</td>
</tr>
<tr>
<td>administrators</td>
<td>is otherwise sent to is specified by the recipient parameter for this action.</td>
</tr>
<tr>
<td>Event</td>
<td>Send an event to users with a connected client</td>
</tr>
<tr>
<td>ImageUpdate</td>
<td>Update the image on the node</td>
</tr>
<tr>
<td>PowerOff</td>
<td>Powers off, hard</td>
</tr>
<tr>
<td>PowerOn</td>
<td>Powers on, hard</td>
</tr>
<tr>
<td>PowerReset</td>
<td>Power reset, hard</td>
</tr>
<tr>
<td>Reboot</td>
<td>Reboot via the system, trying to shut everything down cleanly, and then start up again</td>
</tr>
<tr>
<td>killprocess*</td>
<td>Kills processes listed in STDIN with KILL (-9) signal. Format: killprocess &lt;PID1[,&lt;PID2&gt;,...]&gt;</td>
</tr>
<tr>
<td></td>
<td>This action is designed to work with rogueprocess (page 952)</td>
</tr>
<tr>
<td>remount*</td>
<td>remounts all defined mounts</td>
</tr>
<tr>
<td>testaction*</td>
<td>An action script example for users who would like to create their own scripts. The source has helpful remarks about the environment variables</td>
</tr>
<tr>
<td></td>
<td>that can be used as well as tips on configuring it generally</td>
</tr>
<tr>
<td>Shutdown</td>
<td>Power off via system, trying to shut everything down cleanly</td>
</tr>
<tr>
<td>Undrain node</td>
<td>Allow processes to run on the node from the workload manager</td>
</tr>
<tr>
<td></td>
<td>* standalone scripts, not built-ins.</td>
</tr>
</tbody>
</table>

If running from a head node, the script is in directory: /cm/local/apps/cmd/scripts/actions/
If running from a regular node, the script is in directory: /cm/images/default-image/cm/local/apps/cmd/scripts/actions/

G.4.2 Parameters For A Monitoring Action

The default monitoring actions are listed in section 13.4.4.

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 WIM
Allowed time     no
Disable          yes
Name             Drain
Revision         Active
Run on           Active
Type             DrainAction
```
Out of the full list of default actions, the actions with only the common parameter settings are:

- **Power on**: Powers off the node
- **Power Off**: Powers off the node
- **Power Reset**: 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.
- **Image Update**: Updates the node from the software image
- **Event**: Sends an event to users connected with cmsh or Bright 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
# Workload Manager Configuration Files Updated By CMDaemon

This appendix lists workload manager configuration files changed by CMDaemon, events causing such change, and the file or property changed.

## H.1 Slurm

<table>
<thead>
<tr>
<th>File/Property</th>
<th>Updates What?</th>
<th>Updated During</th>
</tr>
</thead>
<tbody>
<tr>
<td>/cm/shared/apps/slurm/var/etc/</td>
<td>head node</td>
<td>Add/Remove/Update nodes, hostname change</td>
</tr>
<tr>
<td>&lt;Slurm instance name&gt;/slurm.conf</td>
<td></td>
<td></td>
</tr>
<tr>
<td>/cm/shared/apps/slurm/var/etc/</td>
<td>head node</td>
<td>Add/Remove/Update nodes, hostname change</td>
</tr>
<tr>
<td>&lt;Slurm instance name&gt;/slurmd.conf</td>
<td></td>
<td></td>
</tr>
<tr>
<td>/cm/shared/apps/slurm/var/etc/</td>
<td>all nodes</td>
<td>Add/Remove/Update nodes</td>
</tr>
<tr>
<td>&lt;Slurm instance name&gt;/gres.conf</td>
<td></td>
<td></td>
</tr>
<tr>
<td>/cm/shared/apps/slurm/var/etc/</td>
<td>head node</td>
<td>Add/Remove/Update nodes</td>
</tr>
<tr>
<td>&lt;Slurm instance name&gt;/topology.conf</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

## H.2 Univa Grid Engine (UGE)

<table>
<thead>
<tr>
<th>File/Property</th>
<th>Updates What?</th>
<th>Updated During</th>
</tr>
</thead>
<tbody>
<tr>
<td>/cm/shared/apps/uge/var/</td>
<td>head node</td>
<td>hostname/domain change, failover</td>
</tr>
<tr>
<td>&lt;UGE instance name&gt;/common/host_aliases</td>
<td></td>
<td></td>
</tr>
<tr>
<td>/cm/shared/apps/uge/var/</td>
<td>head node</td>
<td>hostname/domain change, failover</td>
</tr>
<tr>
<td>&lt;UGE instance name&gt;/common/qmaster</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

## H.3 PBS Pro/OpenPBS
<table>
<thead>
<tr>
<th>File/Property</th>
<th>Updates What?</th>
<th>Updated During</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>$PBS_CONF_FILE</code></td>
<td>head node, software image</td>
<td>hostname/domain change, failover</td>
</tr>
<tr>
<td><code>/cm/local/apps/&lt;openpbs_orpbspro&gt;/</code></td>
<td>head node</td>
<td>hostname change, failover</td>
</tr>
<tr>
<td><code>var/spool/mom_priv/config</code></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

The default value of `$PBS_CONF_FILE` in the cluster manager is `/cm/local/apps/<openpbs_orpbspro>/var/etc/pbs.conf`

### H.4 LSF

<table>
<thead>
<tr>
<th>File/Property</th>
<th>Updates What?</th>
<th>Updated During</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>$LSF_ENVDIR/lsf.conf</code></td>
<td>head node</td>
<td>hostname/domain change, failover</td>
</tr>
<tr>
<td><code>$LSF_ENVDIR/lsf.cluster.&lt;clustername&gt;</code></td>
<td>head node</td>
<td>add/remove/update nodes</td>
</tr>
<tr>
<td><code>$LSF_ENVDIR/lsf.sudoers</code></td>
<td>head node</td>
<td>hostname/domain change, failover</td>
</tr>
<tr>
<td><code>$LSF_ENVDIR/hosts</code></td>
<td>cloud-director</td>
<td>add/remove/update cloud nodes</td>
</tr>
<tr>
<td><code>$LSF_ENVDIR/lsbatch/&lt;clustername&gt;/configdir/lsb.queues</code></td>
<td>head node</td>
<td>add/remove/update queues</td>
</tr>
<tr>
<td><code>$LSF_ENVDIR/lsbatch/&lt;clustername&gt;/configdir/lsb.hosts</code></td>
<td>head node</td>
<td>add/remove/update nodes</td>
</tr>
</tbody>
</table>

The default value of `$LSF_ENVDIR` in the cluster manager is `/cm/shared/apps/lsf/var/conf/<clustername>`

On each node where the `lsfd` service runs, CMDaemon creates a symlink `/etc/lsf.conf` that points to `$LSF_ENVDIR/lsf.conf`. This is required by LSF daemons.
Changing The LDAP Password

The administrator may wish to change the LDAP root password. This procedure has two steps:

- setting a new password for the LDAP server (section I.1), and
- setting the new password in cmd.conf (section I.2).

It is also a good idea to do some checking afterwards (section I.3).

I.1 Setting A New Password For The LDAP Server

An encrypted password string can be generated as follows:

```
[root@bright92 ~]# module load openldap
[root@bright92 ~]# slappasswd
New password:
Re-enter new password:
SSHAJ/3wy0+IqyAvhh8Q4obL8489CWJ1Rplg
```

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@bright92 ~]# grep ^rootpw /cm/local/apps/openldap/etc/slapd.conf
rootpw SSHAJ/3wy0+IqyAvhh8Q4obL8489CWJ1Rplg
```

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@bright92 ~]# service slapd restart
```

I.2 Setting The New Password In cmd.conf

The new LDAP password (the plain text password that generated the encrypted password after entering the slappasswd command in section I.1) is set in cmd.conf. It is kept as clear text for the entry for the LDAPPass directive (Appendix C):

```
[root@bright92 ~]# grep LDAPPass /cm/local/apps/cmd/etc/cmd.conf
LDAPPass = "Mysecret1dappassw0rd"
```

CMDaemon is then restarted:

```
[root@bright92 ~]# service cmd restart
```
1.3 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@bright92 ~]# ldapsearch -x
  # extended LDIF
  #
  # LDAPv3
  # base <dc=cm,dc=cluster> (default) with scope subtree
  ...  
  ``

- **root cn without a password (this should fail):**

  ```
  [root@bright92 ~]# 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@bright92 ~]#  
  ```

- **root cn with a password (this should work):**

  ```
  [root@bright92 ~]# ldapsearch -x -D 'cn=root,dc=cm,dc=cluster' -w Mysecret1dappassw0rd
  # extended LDIF
  #
  # LDAPv3
  # base <dc=cm,dc=cluster> (default) with scope subtree
  ...  
  ```
This appendix describes authorization tokens available for profiles. Profiles are introduced in Section 6.4:

<table>
<thead>
<tr>
<th>Service and token name</th>
<th>Service: CMAuth</th>
<th>User can...</th>
</tr>
</thead>
<tbody>
<tr>
<td>GET_PROFILE_TOKEN</td>
<td>Retrieve list of profiles and profile properties</td>
<td></td>
</tr>
<tr>
<td>ADD_PROFILE_TOKEN</td>
<td>Add a new profile</td>
<td></td>
</tr>
<tr>
<td>MODIFY_PROFILE_TOKEN</td>
<td>Modify existing user profile</td>
<td></td>
</tr>
<tr>
<td>GET_CM_SERVICES_TOKEN</td>
<td>Get a list of available CMDaemon services</td>
<td></td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Service: CMCeph</th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>GET_CEPH_TOKEN</td>
<td>Get Ceph properties</td>
</tr>
<tr>
<td>ADD_CEPH_TOKEN</td>
<td>Add new Ceph configuration</td>
</tr>
<tr>
<td>UPDATE_CEPH_TOKEN</td>
<td>Update Ceph configuration</td>
</tr>
<tr>
<td>CEPH_KEYS_TOKEN</td>
<td>Manipulate Ceph keys</td>
</tr>
<tr>
<td>CEPH_CLUSTER_SERVICES_TOKEN</td>
<td>List Ceph services</td>
</tr>
<tr>
<td>CEPH_SET_DECOMMISSION_TOKEN</td>
<td>Decommission Ceph nodes</td>
</tr>
<tr>
<td>CEPH_GET_DECOMMISSION_TOKEN</td>
<td>Get decommissioning information</td>
</tr>
<tr>
<td>UPDATE_CEPH_STATE_TOKEN</td>
<td>Update Ceph state</td>
</tr>
<tr>
<td>GENERIC_CEPH_COMMAND_TOKEN</td>
<td>Run generic Ceph command</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Service: CMCert</th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>ISSUE_CERTIFICATE_TOKEN</td>
<td>Accept certificate request and issue signed certificate</td>
</tr>
<tr>
<td>GET_CERTIFICATE_REQUEST_TOKEN</td>
<td>List pending certificate requests</td>
</tr>
<tr>
<td>RECREATE_COMPONENT_CERTIFICATE_TOKEN</td>
<td>Recreate component certificate</td>
</tr>
</tbody>
</table>

...continues
### Table J: List Of Tokens...continued

<table>
<thead>
<tr>
<th>Service and token name</th>
<th>User can...</th>
</tr>
</thead>
<tbody>
<tr>
<td>REMOVE_CERTIFICATE_REQUEST_TOKEN</td>
<td>Cancel certificate request</td>
</tr>
<tr>
<td>GET_CERTIFICATE_TOKEN</td>
<td>Show certificate information</td>
</tr>
<tr>
<td>REVOKE_CERTIFICATE_TOKEN</td>
<td>Revoke a certificate</td>
</tr>
<tr>
<td>UNREVOKE_CERTIFICATE_TOKEN</td>
<td>Unrevoke a revoked certificate</td>
</tr>
<tr>
<td>GET_AUTOSIGN_TOKEN</td>
<td>Get information on certificate auto-signing CMDaemon’s private key</td>
</tr>
<tr>
<td>SET_AUTOSIGN_TOKEN</td>
<td>Enable certificate auto-signing with CMDaemon’s private key</td>
</tr>
</tbody>
</table>

**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 |
| 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_INITIALIZER_MDSUM_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 |
| SET_CLOUDDIRECTOR_TOKEN | Modify the properties of the cloud director |
| 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 |
| GET_CLOUD_STATIC_IPS_TOKEN | Get the static IP list of the cloud nodes |
| GET_CLOUD_VIRTUAL_NETWORK_INTERFACES_TOKEN | Get list of EC2 VPN interfaces (e.g. tun0) |
| SEND_CLOUD_STORAGE_ACTION_TOKEN | Send cloud storage action |
| 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 |
| ALLOCATE_CLOUD_VIRTUAL_NETWORK_INTERFACE_TOKEN | Allocate virtual cloud network interface |
| UNALLOCATE_CLOUD_VIRTUAL_NETWORK_INTERFACE_TOKEN | De-allocate virtual cloud network interface |
| ATTACH_CLOUD_VIRTUAL_NETWORK_INTERFACE_TOKEN | Start a VPN tunnel to a cloud node |

...continues
### Table J: List Of Tokens...continued

<table>
<thead>
<tr>
<th>Service and token name</th>
<th>User can...</th>
</tr>
</thead>
<tbody>
<tr>
<td>DETACH_CLOUD_VIRTUAL_NETWORK_INTERFACE</td>
<td>Stop a VPN tunnel to a cloud node</td>
</tr>
<tr>
<td>ADD_CLOUD_JOB_DESCRIPTION_TOKEN</td>
<td>Add cloud job description</td>
</tr>
<tr>
<td>GET_CLOUD_JOB_DESCRIPTION_TOKEN</td>
<td>Get cloud job description</td>
</tr>
<tr>
<td>GET_ALL_CLOUD_JOB_DESCRIPTION_TOKEN</td>
<td>Get all cloud job description</td>
</tr>
<tr>
<td>SUBMIT_CLOUD_JOB_DESCRIPTION_TOKEN</td>
<td>Submit cloud job description</td>
</tr>
<tr>
<td>UPDATE_CLOUD_JOB_DESCRIPTION_TOKEN</td>
<td>Update cloud job description</td>
</tr>
</tbody>
</table>

### 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 |
| GET_CATEGORY_TOKEN | Get list of categories |
| ADD_CATEGORY_TOKEN | Create new category |
| UPDATE_CATEGORY_TOKEN | Update a category property |
| GET_NODEGROUP_TOKEN | Get list of nodegroups |
| ADD_NODEGROUP_TOKEN | Add a new nodegroup |
| UPDATE_NODEGROUP_TOKEN | Update nodegroup properties (e.g. add a new member node) |
| 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_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 |
| SHUTDOWN_NODE_TOKEN | Shutdown a remote node managed by CMDaemon |
| REBOOT_NODE_TOKEN | Reboot a remote node |
| FORCE_RECONNECT_TOKEN | Force remote client to reconnect (RPC API) |
| NODE_IDENTIFY_TOKEN | Identify a node (RPC API, used by node installer) |

...continues
Table J: List Of Tokens...continued

<table>
<thead>
<tr>
<th>Service and token name</th>
<th>User can...</th>
</tr>
</thead>
<tbody>
<tr>
<td>NODE_GET_MOUNTPOINTS_TOKEN</td>
<td>Get list of mountpoints defined for a node</td>
</tr>
<tr>
<td>PPING_TOKEN</td>
<td>Run parallel ping</td>
</tr>
<tr>
<td>BURN_STATUS_TOKEN</td>
<td>Get burn status</td>
</tr>
<tr>
<td>GET_BURN_LOG_TOKEN</td>
<td>Retrieve burn log</td>
</tr>
<tr>
<td>GET_SYNC_LOG_TOKEN</td>
<td>Get rsync provisioning log</td>
</tr>
<tr>
<td>GET_PORT_BY_MAC_TOKEN</td>
<td>Determine to which switch port a given MAC is connected to.</td>
</tr>
<tr>
<td>SYSINFO_COLLECTOR_TOKEN</td>
<td>Get information about a node (executes dmidecode)</td>
</tr>
<tr>
<td>UPDATE_CONFIG_TOKEN</td>
<td>Update node configuration</td>
</tr>
<tr>
<td>GET_LIVE_UCS_TOKEN</td>
<td>Get UCS live status</td>
</tr>
<tr>
<td>SET_LIVE_UCS_TOKEN</td>
<td>Set UCS live</td>
</tr>
<tr>
<td>VALIDATE_UCS_TOKEN</td>
<td>Validate UCS configuration</td>
</tr>
<tr>
<td>START_KVM_UCS_TOKEN</td>
<td>Start UCS KVM console</td>
</tr>
<tr>
<td>GET_UCS_LOG_TOKEN</td>
<td>Retrieve UCS log</td>
</tr>
<tr>
<td>CLEAR_UCS_LOG_TOKEN</td>
<td>Clear Cisco UCS log</td>
</tr>
<tr>
<td>CONVERT_XML_TO_UCS_TOKEN</td>
<td>Parse and serialize XML UCS configuration so that CM-Daemon can use it</td>
</tr>
<tr>
<td>GET_EXCLUDE_LIST_TOKEN</td>
<td>Retrieve the various exclude lists</td>
</tr>
<tr>
<td>INSTALLER_REBOOT_REQUIRED_TOKEN</td>
<td>Set restart-required flag</td>
</tr>
<tr>
<td>(typically set by CMDaemon)</td>
<td></td>
</tr>
<tr>
<td>ADD_REMOTE_NODE_INSTALLER_INTERACTION_TOKEN</td>
<td>Add a node-installer interaction (Used by CMDaemon)</td>
</tr>
<tr>
<td>REMOVE_REMOTE_NODE_INSTALLER_INTERACTION_TOKEN</td>
<td>Remove a node installer interaction</td>
</tr>
<tr>
<td>INSTALLER_INTERACTION_TOKEN</td>
<td></td>
</tr>
<tr>
<td>GET_REMOTE_NODE_INSTALLER_INTERACTIONS_TOKEN</td>
<td>Get list of pending installer interactions (e.g. confirm full provisioning)</td>
</tr>
<tr>
<td>UPDATE_REMOTE_NODE_INSTALLER_INTERACTIONS_TOKEN</td>
<td>Update installer</td>
</tr>
<tr>
<td>GET_CLUSTAT</td>
<td>Get cluster status (RPC API, Internal)</td>
</tr>
<tr>
<td>SET_CLUSTAT</td>
<td>Set cluster status (RPC API, Internal)</td>
</tr>
</tbody>
</table>

Service: CMGui

| GET_CLUSTER_OVERVIEW_TOKEN                   | Get 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                                  |
| TOKEN                                        |                                                   |
| GET_CEPH_OVERVIEW_TOKEN                      | Get Ceph overview                                |
| OVERVIEW_TOKEN                               | overview                                         |

...continues
<table>
<thead>
<tr>
<th>Service and token name</th>
<th>User can...</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Service: CMJob</strong></td>
<td></td>
</tr>
<tr>
<td>GET_JOB_TOKEN</td>
<td>Get list of jobs that are currently running</td>
</tr>
<tr>
<td>HOLD_JOB_TOKEN</td>
<td>Place a job on hold</td>
</tr>
<tr>
<td>SUSPEND_JOB_TOKEN</td>
<td>Suspend a job</td>
</tr>
<tr>
<td>RESUME_JOB_TOKEN</td>
<td>Resume suspended job</td>
</tr>
<tr>
<td>RELEASE_JOB_TOKEN</td>
<td>Release a held job</td>
</tr>
<tr>
<td>UPDATE_JOB_TOKEN</td>
<td>Update job run-timer parameters</td>
</tr>
<tr>
<td>SUBMIT_JOB_TOKEN</td>
<td>Submit a job using JSON</td>
</tr>
<tr>
<td>GET_JOBQUEUE_TOKEN</td>
<td>Retrieve list of job queues and properties</td>
</tr>
<tr>
<td>UPDATE_JOBQUEUE_TOKEN</td>
<td>Modify job queues</td>
</tr>
<tr>
<td>ADD_JOBQUEUE_TOKEN</td>
<td>Add a new job queue</td>
</tr>
<tr>
<td>GET_PE_TOKEN</td>
<td>Get list of SGE parallel environments</td>
</tr>
<tr>
<td>DRAIN_TOKEN</td>
<td>Drain a node</td>
</tr>
<tr>
<td>DRAIN_OVERVIEW_TOKEN</td>
<td>Obtain list of drained nodes</td>
</tr>
<tr>
<td><strong>Service: CMMain</strong></td>
<td></td>
</tr>
<tr>
<td>GET_LICENSE_INFO_TOKEN</td>
<td>Retrieve information about BCM license</td>
</tr>
<tr>
<td>GET_VERSION_TOKEN</td>
<td>Get CMDaemon version and revision</td>
</tr>
<tr>
<td>GET_SERVER_STATUS_TOKEN</td>
<td>Head node status (e.g. ACTIVE, BECOMEACTIVE etc.)</td>
</tr>
<tr>
<td>GET_CLUSTER_SETUP_TOKEN</td>
<td>Get cluster configuration</td>
</tr>
<tr>
<td>PING_TOKEN</td>
<td>TCP SYN ping managed devices</td>
</tr>
<tr>
<td>PCOPY_TOKEN</td>
<td>Copy a specified file in parallel to a list of nodes</td>
</tr>
<tr>
<td>READ_FILE_TOKEN</td>
<td>Read a text file. The text file will be serialized as a JSON object</td>
</tr>
<tr>
<td>SAVE_FILE_TOKEN</td>
<td>Save a file on a remote node</td>
</tr>
<tr>
<td>UPDATE_SELF_TOKEN</td>
<td>Update a category property</td>
</tr>
<tr>
<td>CMDAEMON_FAILOVER_TOKEN</td>
<td>Set CMDaemon failover condition achieved</td>
</tr>
<tr>
<td>CMDAEMON_QUORUM_TOKEN</td>
<td>Set CMDaemon quorum achieved</td>
</tr>
<tr>
<td>GENERIC_CALL_TOKEN</td>
<td>Make a generic call</td>
</tr>
<tr>
<td>REPORT_CRITICAL_ERROR_TOKEN</td>
<td>View critical error report</td>
</tr>
<tr>
<td>SET_SERVICESTATE_TOKEN</td>
<td>Set the state of a service</td>
</tr>
<tr>
<td>GET_SERVICESTATE_TOKEN</td>
<td>Get the state of a service</td>
</tr>
<tr>
<td>GET_BACKGROUND_TASKS_TOKEN</td>
<td>Get background tasks</td>
</tr>
<tr>
<td>CANCEL_BACKGROUND_TASKS_TOKEN</td>
<td>Cancel background tasks</td>
</tr>
<tr>
<td>CALL_EXTENSION_TOKEN</td>
<td></td>
</tr>
<tr>
<td><strong>Service: CMMon</strong></td>
<td></td>
</tr>
<tr>
<td>GET_MONCONF_TOKEN</td>
<td>Get monitoring configuration settings</td>
</tr>
<tr>
<td>UPDATE_MONCONF_TOKEN</td>
<td>Update monitoring configuration settings</td>
</tr>
</tbody>
</table>

...continues
Table J: List Of Tokens...continued

<table>
<thead>
<tr>
<th>Service and token name</th>
<th>User can...</th>
</tr>
</thead>
<tbody>
<tr>
<td>GET_METRIC_TOKEN</td>
<td>Get metric settings</td>
</tr>
<tr>
<td>UPDATE_METRIC_TOKEN</td>
<td>Update metric settings</td>
</tr>
<tr>
<td>ADD_METRIC_TOKEN</td>
<td>Add metric settings</td>
</tr>
<tr>
<td>GET_HEALTHCHECK_TOKEN</td>
<td>Get health check settings</td>
</tr>
<tr>
<td>UPDATE_HEALTHCHECK_TOKEN</td>
<td>Update health check settings</td>
</tr>
<tr>
<td>ADD_HEALTHCHECK_TOKEN</td>
<td>Add health check settings</td>
</tr>
<tr>
<td>GET_THRESHOLDACTION_TOKEN</td>
<td>Get threshold action settings</td>
</tr>
<tr>
<td>UPDATE_THRESHOLDACTION_TOKEN</td>
<td>Update threshold action settings</td>
</tr>
<tr>
<td>ADD_THRESHOLDACTION_TOKEN</td>
<td>Add threshold action settings</td>
</tr>
<tr>
<td>GET_MONITORINGDATA_TOKEN</td>
<td>Get monitoring data settings</td>
</tr>
<tr>
<td>SIGNAL_THRESHOLDEXCEEDED_TOKEN</td>
<td>See if signal threshold exceeded</td>
</tr>
<tr>
<td>MONITORINGINTERNAL_TOKEN</td>
<td>Set up internal monitoring</td>
</tr>
<tr>
<td>PREJOB_TOKEN</td>
<td>Set up prejob check</td>
</tr>
<tr>
<td>FAKE_MONITORING_TOKEN</td>
<td>Set up fake monitoring</td>
</tr>
<tr>
<td>ONDEMAND_TOKEN</td>
<td>Sample on demand</td>
</tr>
<tr>
<td>ONDEMANDRESULT_TOKEN</td>
<td>Read sample results on demand.</td>
</tr>
<tr>
<td>GET_METRICCLASS_TOKEN</td>
<td>Get metric class settings</td>
</tr>
</tbody>
</table>

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 |
GETSOFTWAREIMAGE_TOKEN | Get softwareimage settings |
ADDSOFTWAREIMAGE_TOKEN | Add softwareimage settings |
UPDATESOFTWAREIMAGE_TOKEN | Update softwareimage settings |
REMOTESOFTWAREIMAGE_TOKEN | Remove softwareimage settings |
UPDATEPROVISIONERS_TOKEN | Update provisioners settings |
UPDATEPROVISIONINGNODE_TOKEN | Update provisioning node |
CMDAEMONFAILOVERSTATUS_TOKEN | Obtain status of failover |

...continues
### Table J: List Of Tokens...continued

<table>
<thead>
<tr>
<th>Service and token name</th>
<th>User can...</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Service: CMProc</strong></td>
<td></td>
</tr>
<tr>
<td>GET_PROCESS_TOKEN</td>
<td>Retrieve list of processes that are currently running on a device managed by CMDaemon</td>
</tr>
<tr>
<td>GET_SHARED_MEM_TOKEN</td>
<td>Get shared memory</td>
</tr>
<tr>
<td>GET_SEMAPHORE_TOKEN</td>
<td>Get semaphore</td>
</tr>
<tr>
<td>GET_MSGQUEUE_TOKEN</td>
<td>Get message queue status</td>
</tr>
<tr>
<td>CLEAN_IPC_TOKEN</td>
<td>Clear IPC state</td>
</tr>
<tr>
<td>SEND_SIGNAL_TOKEN</td>
<td>Send signal to a process</td>
</tr>
<tr>
<td>START_SHELL_TOKEN</td>
<td>Start SSH session</td>
</tr>
<tr>
<td>START_MINICOM_TOKEN</td>
<td>Start minicom serial session</td>
</tr>
<tr>
<td>EXEC_COMMAND_TOKEN</td>
<td>Execute a command on a head node</td>
</tr>
<tr>
<td>NODE_EXEC_COMMAND_TOKEN</td>
<td>Remotely execute a command on a compute node</td>
</tr>
<tr>
<td>MAINTENANCE_TOKEN</td>
<td>Execute a maintenance command (used by CMDaemon)</td>
</tr>
<tr>
<td>EXEC_INTERNAL_COMMAND_TOKEN</td>
<td>Execute internal command (defined in the source code, RPC API, internal)</td>
</tr>
<tr>
<td><strong>Service: CMProv</strong></td>
<td></td>
</tr>
<tr>
<td>GET_FSPART_TOKEN</td>
<td>Get FSPart (internal)</td>
</tr>
<tr>
<td>ADD_FSPART_TOKEN</td>
<td>Set FSPart (internal)</td>
</tr>
<tr>
<td>UPDATE_FSPART_TOKEN</td>
<td>Update FSPart (internal)</td>
</tr>
<tr>
<td>REMOVE_FSPART_TOKEN</td>
<td>Remove FSPart (internal)</td>
</tr>
<tr>
<td>RUN_PROVISIONINGPROCESSORJOB_TOKEN</td>
<td>Start and run a provisioning job (nodes with a provisioning role)</td>
</tr>
<tr>
<td>UPDATE_PROVISIONINGPROCESSORJOB_TOKEN</td>
<td>Update status of running provisioning jobs (CMDaemon)</td>
</tr>
<tr>
<td>REQUEST_PROVISIONING_TOKEN</td>
<td>Request provisioning (nodes with a provisioning role)</td>
</tr>
<tr>
<td>MANAGE_RSYNC_DAEMON_TOKEN</td>
<td>Manage the rsync process (CMDaemon)</td>
</tr>
<tr>
<td>IMAGEUPDATE_TOKEN</td>
<td>Send image changes to nodes</td>
</tr>
<tr>
<td>UPDATEPROVISIONERS_TOKEN</td>
<td>Synchronize software images across provisioning systems (requires at least two provisioners)</td>
</tr>
<tr>
<td>PROVISIONERS_STATUS_TOKEN</td>
<td>Check status of provisioners e.g. images are in sync</td>
</tr>
<tr>
<td>CANCEL_PROVISIONING_REQUEST_TOKEN</td>
<td>Cancel provisioning request</td>
</tr>
<tr>
<td>GRAB_IMAGEUPDATE_TOKEN</td>
<td>Grab changes from node to software image and vice versa</td>
</tr>
<tr>
<td><strong>Service: CMServ</strong></td>
<td></td>
</tr>
<tr>
<td>GET_OSSERVICE_TOKEN</td>
<td>Get system service information</td>
</tr>
<tr>
<td>START_OSSERVICE_TOKEN</td>
<td>Start system services (service foo start)</td>
</tr>
<tr>
<td>STOP_OSSERVICE_TOKEN</td>
<td>Stop system services</td>
</tr>
</tbody>
</table>

...continues
### Table J: List Of Tokens...continued

<table>
<thead>
<tr>
<th>Service and token name</th>
<th>User can...</th>
</tr>
</thead>
<tbody>
<tr>
<td>CALLINIT_OSSERVICE_TOKEN</td>
<td>Call init (useful for the node-installer itself)</td>
</tr>
<tr>
<td>RESTART_OSSERVICE_TOKEN</td>
<td>Restart system services</td>
</tr>
<tr>
<td>RELOAD_OSSERVICE_TOKEN</td>
<td>Reload system services</td>
</tr>
<tr>
<td>RESET_OSSERVICE_TOKEN</td>
<td>Reset system services</td>
</tr>
</tbody>
</table>

**Service: CMSession**

<table>
<thead>
<tr>
<th>Service and token name</th>
<th>User can...</th>
</tr>
</thead>
<tbody>
<tr>
<td>GET_SESSION_TOKEN</td>
<td>Retrieve session information</td>
</tr>
<tr>
<td>REGISTER_NODE_SESSION_TOKEN</td>
<td>Register new nodes in a special CMDaemon session (node-installer)</td>
</tr>
<tr>
<td>END_SESSION_TOKEN</td>
<td>Terminate sessions</td>
</tr>
<tr>
<td>HANDLE_EVENT_TOKEN</td>
<td>Handle events</td>
</tr>
<tr>
<td>GET_BROADCAST_EVENTS_TOKEN</td>
<td>Receive broadcast events</td>
</tr>
</tbody>
</table>

**Service: CMUser**

<table>
<thead>
<tr>
<th>Service and token name</th>
<th>User can...</th>
</tr>
</thead>
<tbody>
<tr>
<td>GET_USER_TOKEN</td>
<td>Retrieve user information</td>
</tr>
<tr>
<td>ADD_USER_TOKEN</td>
<td>Add a new LDAP user</td>
</tr>
<tr>
<td>UPDATE_USER_TOKEN</td>
<td>Modify an existing LDAP user</td>
</tr>
<tr>
<td>GET_GROUP_TOKEN</td>
<td>Retrieve group information</td>
</tr>
<tr>
<td>ADD_GROUP_TOKEN</td>
<td>Add a new LDAP group</td>
</tr>
<tr>
<td>UPDATE_GROUP_TOKEN</td>
<td>Modify an existing LDAP group</td>
</tr>
</tbody>
</table>
Understanding Consolidation

K.1 Introduction
Consolidation is discussed in the sections on using consolidation in the Monitoring chapter (sections 13.4.3 and 13.5.2).

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 600) may not appeal to people. There are many who would like an explanation that may be more intuitive, even if it is less rigorous.

Therefore, in this section a more informal and visual approach is taken to explain consolidation.

K.2 What Is Consolidation?
Consolidation is the compression of data, for data values that have been measured over a fixed interval.

The compression is nothing particularly sophisticated. It is carried out by using some simple mathematical functions to the data points: the average, the maximum, or the minimum.

K.3 Raw Data And Consolidation
Suppose raw data is sampled every 2 minutes.

And the raw data values are consolidated every 10 minutes.

A visual representation of the data values available to the system is:

--- time --->
raw: | | | | | | | | | | | | | | | |
consolidated: | | | | | | | | | | | | | | |

Here, every "|" indicates a data point, so that the visual shows 5 times as many raw data points as consolidated data values.

In the preceding visual it makes no sense to use consolidated data since the data values for raw data and consolidated data overlap. I.e., the more accurate raw data values exist for the entire period.

As time passes, the intention is to start dropping old raw data, to save space on the disk.

For example, for the first 20 minutes in the following visual, there are no longer raw data values available:

Example

--- time --->
raw: | | | | | | | | | | | | | | | |
consolidated: | | | | | | | | | | | | | | | |
But the consolidated data points for this period are still available to the system. When the data values are plotted in Bright View graphs, periods without raw data values automatically have consolidated data values used.

So a combination of both data sources is used, which can be visually represented with:

Example

```
--- time --->
plot:   |   |   |   |   |   |   |   |   |   |   |   |   |
```

That behavior holds true for cmsh too.

The behavior illustrated in the last visual assumes that the cluster has been UP for long enough that raw data is being dropped.

In this case, “long enough” means at least 7 days.

However, because RLE (Run Length Encoding) is used to compress the sampled monitoring data values on disk, this minimal “long enough” time can be (much) longer than 7 days. It depends on how much the measurable that is being sampled is changing as each sample is taken. If it is not changing, then RLE can compress over a longer time period.

For example, if a node has been up and reachable without issues for 1000 days, then the ssh2node health check raw data values would be PASS over that 1000 days. For the period from now to 7 days ago, the raw data values of PASS are kept as they are for now. However, for the period from 7 days ago to 1000 days ago, consolidation on the unchanging raw values means that only two values, namely the PASS value of 1000 days and the PASS value of 7 days ago, need to be retained, in order to have a totally accurate record of what the values were in that period.

On the other hand, the forks metric changes very quickly, and thus can do little RLE compression. That makes it a good choice for demonstrating the kind of output that the preceding visuals imply.

**K.4 A Demonstration Of The Output**

So, as a demonstration, the last 7 days for forks are now shown, with the data values in the middle elided:

Example

```
[bright92->device[bright92]]% 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
<table>
<thead>
<tr>
<th>Timestamp</th>
<th>Value</th>
<th>Info</th>
</tr>
</thead>
<tbody>
<tr>
<td>2019/01/07 11:39:06</td>
<td>2.65704</td>
<td>processes/s</td>
</tr>
<tr>
<td>2019/01/07 12:30:00</td>
<td>2.60111</td>
<td>processes/s</td>
</tr>
<tr>
<td>2019/01/07 13:30:00</td>
<td>2.58328</td>
<td>processes/s</td>
</tr>
<tr>
<td>...</td>
<td></td>
<td></td>
</tr>
<tr>
<td>2019/01/07 18:09:06</td>
<td>2.59586</td>
<td>processes/s</td>
</tr>
<tr>
<td>2019/01/08 13:00:00</td>
<td>2.58953</td>
<td>processes/s</td>
</tr>
<tr>
<td>2019/01/09 06:06:06</td>
<td>2.58854</td>
<td>processes/s</td>
</tr>
<tr>
<td>...</td>
<td></td>
<td></td>
</tr>
<tr>
<td>2019/01/08 11:15:12.194</td>
<td>2.59113</td>
<td>processes/s</td>
</tr>
</tbody>
</table>
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 (bright92) 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:

```bash
[root@bright92 ~]# cat /cm/shared/fm.sh
#!/bin/bash

echo $((RANDOM%100))

echo "Sampled on $(hostname) for $CMD_HOSTNAME" >&3
```

The preceding script outputs a random number, along with the host it is being run on ($hostname), and the host the metric is targeting ($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 multiplexer nodes can have their samples displayed as output using the samplenow command (section 13.6.2, page 626) of cmsh.
### L.1 Data Producers: Default Configuration For Running And Sampling

If there is no configuration defined for the data producer in the filters or multiplexers for that data producer, then each node runs a data producer on itself, and that data producer targets the node that it is running on.

For example, the existing `dmesg` data producer comprises the `dmesg` health check (section G.2.1) and by default has no filter or multiplexer defined for it. If an attempt is made to list any filter or multiplexer for `dmesg`, then by default there is no content under the table headings:

```text
[root@bright92 ~]# cmsh
[bright92]# monitoring setup
[bright92->monitoring->setup]# nodeexecutionfilters dmesg; list; ...
<table>
<thead>
<tr>
<th>Type</th>
<th>Name (key)</th>
<th>Filter</th>
<th>Filter operation</th>
</tr>
</thead>
</table>
[bright92->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:

```text
[bright92->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**

```text
[bright92->monitoring->setup]# list -f name,nodeexecutionfilters,executionmultiplexer | more
<table>
<thead>
<tr>
<th>name (key)</th>
<th>nodeexecutionfilters</th>
<th>executionmultiplexer</th>
</tr>
</thead>
<tbody>
<tr>
<td>AggregateNode</td>
<td>&lt;1 in submode&gt;</td>
<td>&lt;1 in submode&gt;</td>
</tr>
<tr>
<td>AggregatePDU</td>
<td>&lt;1 in submode&gt;</td>
<td>&lt;1 in submode&gt;</td>
</tr>
<tr>
<td>AlertLevel</td>
<td>&lt;1 in submode&gt;</td>
<td>&lt;1 in submode&gt;</td>
</tr>
<tr>
<td>CMDaemonState</td>
<td>&lt;0 in submode&gt;</td>
<td>&lt;0 in submode&gt;</td>
</tr>
<tr>
<td>Cassandra</td>
<td>&lt;1 in submode&gt;</td>
<td>&lt;0 in submode&gt;</td>
</tr>
<tr>
<td>ClusterTotal</td>
<td>&lt;1 in submode&gt;</td>
<td>&lt;0 in submode&gt;</td>
</tr>
<tr>
<td>DeviceState</td>
<td>&lt;1 in submode&gt;</td>
<td>&lt;0 in submode&gt;</td>
</tr>
</tbody>
</table>

```

#### L.1.1 Nodes That Data Producers Are Running On By Default—The `nodes` Command

The `nodes` command shows which nodes the data producer runs on. By default, the data producer runs on all nodes, when nothing has been set explicitly, because each node runs the data producer for itself:

```text
[bright92->monitoring->setup]# nodes dmesg
node001..node004,bright92
```

#### L.1.2 Nodes That Data Producers Target By Default—The `samplenow` Command

Nodes where samples are being obtained at can be seen using the `samplenow` command for the specified nodes.

Again, by default, each node is a target, because the target is same node that the `dmesg` data producer runs on:
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,bright92` for this reference cluster.

### L.2 Data Producers: Configuration For Running And Targeting

Filters and multiplexers define which nodes run the data producers, and which nodes are the targets for measurables.

The `fm.sh` script introduced on page 975 can be used to define several custom metrics according to what nodes run the script and what nodes are targeted by the script.

#### L.2.1 Custom Metrics From The `fm.sh` Custom Script

Custom data producers of type `metric` are created in this section. These data producers comprise custom metrics, which are now set up with varying filtering and multiplexing definitions, to illustrate how the definitions work.

**The Metric `all_for_self`**

The metric from the data producer `all_for_self` can be set up with no filtering or multiplexing defined, as follows:

```bash
[bright92->monitoring->setup]% add metric all_for_self
[bright92->monitoring->setup][all_for_self]% set consolidator none
[bright92->monitoring->setup][all_for_self]% set script /cm/shared/fm.sh
[bright92->monitoring->setup][all_for_self]% set class Test
[bright92->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 Bright 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:

```bash
[bright92->monitoring->setup][all_for_self]% exit
[bright92->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
bright92_ all_for_self Test 63_ 0.144s_ Sampled on bright92 for bright92
```
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:

```plaintext
# on node001,node002 for itself
[bright92->monitoring->setup]% add metric some_for_self
[bright92->monitoring->setup*[some_for_self*]]% set consolidator none
[bright92->monitoring->setup*[some_for_self*]]% set class Test
[bright92->monitoring->setup*[some_for_self*]]% set script /cm/shared/fm.sh
[bright92->...*[some_for_self*]->nodeexecutionfilters
[bright92->...*[some_for_self*]->nodeexecutionfilters]% nodeexecutionfilters
[bright92->...*[some_for_self*]->nodeexecutionfilters]% add node some_nodes
[bright92->...*[some_for_self*]->nodeexecutionfilters*[some_nodes*]]% set nodes node001 node002
[bright92->...*[some_for_self*]->nodeexecutionfilters*[some_nodes*]]% show
Parameter Value
-------------------------------- ------------------------------------------------
Filter operation Include
Name some_nodes
Nodes node001,node002
Revision
Type Node
[bright92->...*[some_for_self*]->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:

```plaintext
[bright92->...*[some_for_self*]->nodeexecutionfilters*[some_nodes*]]% ..; ..; ..
[bright92->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:

```plaintext
# on active head for node003,node004
[bright92->monitoring->setup]% add metric from_head_for_some_others
[bright92->monitoring->setup*[from_head_for_some_others*]]% set consolidator none
[bright92->monitoring->setup*[from_head_for_some_others*]]% set script /cm/shared/fm.sh
[bright92->monitoring->setup*[from_head_for_some_others*]]% set class Test
[bright92->...*[from_head_for_some_others*]->nodeexecutionfilters
[bright92->...*[from_head_for_some_others*]->nodeexecutionfilters]% nodeexecutionfilters
[bright92->...*[from_head_for_some_others*]->nodeexecutionfilters]% active
Added active resource filter
[bright92->...*[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
[bright92->monitoring->setup*[from_head_for_some_others*]->nodeexecutionfilters]% ..
[bright92->...*[from_head_for_some_others*]]% executionmultiplexers
```
L.3 Replacing A Resource With An Explicit Node Specification

[...]

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**

[...]

**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:

[...]

The `nodes` command confirms that the head node, bright92 is the filtered node, that is, the only node running the metric:

[...]

L.3 Replacing A Resource With An Explicit Node Specification

Within the filter Active head node, associated with the metric from_head_for_some_others, the resource object Active can be replaced with a node object instead, if the node is defined as being bright92. 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:

[...]

<table>
<thead>
<tr>
<th>Type</th>
<th>Name (key)</th>
<th>Filter</th>
<th>Filter operation</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
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:

```bash
[bright92->monitoring->setup]% device samplenow -t node from_head_for_some_others
```

<table>
<thead>
<tr>
<th>Entity</th>
<th>Measurable</th>
<th>Type</th>
<th>Value</th>
<th>Age</th>
<th>Info</th>
</tr>
</thead>
<tbody>
<tr>
<td>node003</td>
<td>from_head_for_some_others</td>
<td>Test</td>
<td>11</td>
<td>0.081s</td>
<td>Sampled on bright92 for node003</td>
</tr>
<tr>
<td>node004</td>
<td>from_head_for_some_others</td>
<td>Test</td>
<td>12</td>
<td>0.07s</td>
<td>Sampled on bright92 for node004</td>
</tr>
</tbody>
</table>

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.

```bash
L.4 Excessive Sampling

If another node is appended to the node in the filter, then a warning comes up.

```bash
[bright92->monitoring->setup]% nodeexecutionfilters from_head_for_some_others
[bright92->...[from_head_for_some_others]->nodeexecutionfilters]% append head nodes node003
==----------------------------- 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
```

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/multiplex system according to what the administrator has specified. The `nodes` and `samplingnow` commands now show:

```plaintext
[bright92->...[from_head_for_some_others]->nodeexecutionfilters]%...;
[bright92->monitoring->setup]% device sampling -t node from_head_for_some_others
```

<table>
<thead>
<tr>
<th>Entity</th>
<th>Measurable</th>
<th>Type</th>
<th>Value</th>
<th>Age</th>
<th>Info</th>
</tr>
</thead>
<tbody>
<tr>
<td>node003</td>
<td>from_head_for_some_others</td>
<td>Test</td>
<td>3</td>
<td>0.151s</td>
<td>Sampled on bright92 for node003</td>
</tr>
<tr>
<td>node003</td>
<td>from_head_for_some_others</td>
<td>Test</td>
<td>76</td>
<td>0.08s</td>
<td>Sampled on node003 for node003</td>
</tr>
<tr>
<td>node004</td>
<td>from_head_for_some_others</td>
<td>Test</td>
<td>25</td>
<td>0.08s</td>
<td>Sampled on node003 for node004</td>
</tr>
<tr>
<td>node004</td>
<td>from_head_for_some_others</td>
<td>Test</td>
<td>88</td>
<td>0.151s</td>
<td>Sampled on bright92 for node004</td>
</tr>
</tbody>
</table>

```
[bright92->monitoring->setup]% nodes from_head_for_some_others
node003,bright92
```

The node node003 is now doing what the head node is, sampling the same targets, which is typically a bad idea. However the behavior is indeed as expected for this particular configuration.

Whether running a particular configuration is actually wise, is up to the administrator—but in any case the filter/multiplex system allows plenty of abuse of this kind.

## L.5 Not Just For Nodes

Nodes are what node execution filters and execution multiplexers run on. However, sometimes it is more convenient to execute based on other types.

The possible types can be listed with tab-completion suggestions when adding a node execution filter or an execution multiplier:

```
[bright92->monitoring->setup[<data producer>]->nodeexecutionfilters]% add<TAB><TAB>
category lua node overlay resource type
```

### L.6 Lua Node Execution Filters

Lua ([https://www.lua.org/](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**

```
[bright92->monitoring->setup[<data producer>]->nodeexecutionfilters]% add lua lua-filter
[bright92->...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**

```lua
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**
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**

```bash
[root@bright92 ~]# cat node001.lua
self = {}
self.hostname = "node001"
self.category = "default"
```

```bash
[root@bright92 ~]# cat bright92.lua
self = {}
self.hostname = "bright92"
```

This is in addition to the original filter script:

**Example**

```bash
[root@bright92 ~]# 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**

```bash
[root@bright92 ~]# lua
Lua 5.1.4 Copyright (C) 1994-2015 Lua.org, PUC-Rio
> dofile('bright92.lua')
false
> print(dofile('filter.lua'))
false
> dofile('node001.lua')
true
> 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@bright92 ~]# cat ib-slurm-filter.lua
if self == nil then
    return false
end

on_ib_network = false
```
for index, interface in ipairs(self.interface) do
  on_ib_network = on_ib_network or (interface.network == "ibnet")
done

slurm_client = false
for index, role in ipairs(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@bright92 ~]# cat file-check-filter.lua
if self == nil then
  return false
end

function file_exists(name)
  local f = io.open(name, "r")
  if f ~= nil then
    io.close(f)
    return true
  else
    return false
  end
end

return file_exists(string.format("/opt/filter/%s", self.hostname))

It is important to understand that this Lua script is evaluated for all nodes, on the active head node. The Lua script should therefore be fast, and return within a few milliseconds.
M.1 Modes

A 3-level tree of the modes in cmsh is:

|-- beegfs
|-- category
| |-- biossettings
| |-- bmcsettings
| |-- fexports
| |-- fmounts
| |-- gpussettings
| |-- kernelmodules
| |-- roles
| | |-- advancedsettings
| | |-- cgroups
| | |-- commsettings
| | |-- configs
| | |-- configurations
| | |-- connectionsettings
| | |-- engines
| | |-- environments
| | |-- excluselistsnippets
| | |-- genericresources
| | |-- interfaces
| | |-- logsettings
| | |-- momsettings
| | |-- nginxreverseproxy
| | |-- nodecustomizations
| | |-- openports
| | |-- osdconfigurations
| | |-- policies
| | |-- provisioningassociations
| | |-- resourceproviders
| | |-- spawner
| | |-- storagedrivers
| | |-- storagebackends
| | `-- zones
| |-- selinuxsettings
| |-- services
| `-- staticroutes
|-- ceph
 `-- pools
A Tree View Of cmsh

|-- 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
  |-- gpuzsettings
  |-- interfaces
  |-- kernelmodules
  |-- keyvaluesettings
  |-- roles
    |-- advancedsettings
    |-- cgroups
    |-- commsettings
    |-- configs
    |-- configurations
    `-- connectionsettings
    |-- engines